8XC196MC

USER'S MANUAL

intel

Order Number: 272181-001

intel

LITERATURE

To order Intel literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your ***local*** sales office or distributor.

INTEL LITERATURE SALES
P.O. Box 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725
This 800 number is for external customers only.

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information. All handbooks can be ordered individually, and most are available in a pre-packaged set in the U.S. and Canada.

TITLE	INTEL ORDER NUMBER	ISBN
SET OF TEN HANDBOOKS (Available in U.S. and Canada)	**231003**	**N/A**
CONTENTS LISTED BELOW FOR INDIVIDUAL ORDERING:		
EMBEDDED CONTROLLERS & PROCESSORS (2 volume set)	270645	1-55512-140-3
MEMORY PRODUCTS	210830	1-55512-144-6
MICROCOMMUNICATIONS	231658	1-55512-148-9
MICROCOMPUTER PRODUCTS	280407	1-55512-143-8
MICROPROCESSORS	230843	1-55512-150-0
MULTIMEDIA & SUPERCOMPUTING PROCESSORS	272084	1-55512-149-7
PACKAGING	240800	1-55512-145-4
PERIPHERAL COMPONENTS	296467	1-55512-146-2
PRODUCT OVERVIEW (A guide to Intel Architectures and Applications)	210846	1-55512-142-x
PROGRAMMABLE LOGIC	296083	1-55512-147-0
ADDITIONAL LITERATURE: (Not included in handbook set)		
AUTOMOTIVE HANDBOOK	231792	1-55512-125-x
COMPONENTS QUALITY/RELIABILITY	210997	1-55512-132-2
CUSTOMER LITERATURE GUIDE	210620	N/A
EMBEDDED APPLICATIONS	270648	1-55512-123-3
INTERNATIONAL LITERATURE GUIDE (Available in Europe only)	E00029	N/A
MILITARY HANDBOOK (2 volume set)	210461	1-55512-126-8
SYSTEMS QUALITY/RELIABILITY	231762	1-55512-046-6
HANDBOOK DIRECTORY (Index of all data sheets contained in the handbooks)	241197	N/A

intel

8XC196MC
USER'S MANUAL

1992

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

376, Above, ActionMedia, BITBUS, Code Builder, DeskWare, Digital Studio, DVI, EtherExpress, ETOX, ExCA, FaxBACK, Grand Challenge, i, i287, i386, i387, i486, i487, i750, i860, i960, ICE, iLBX, Inboard, Intel, Intel287, Intel386, Intel387, Intel486, Intel487, intel inside., Intellec, iPSC, iRMX, iSBC, iSBX, iWarp, LANPrint, LANSelect, LANShell, LANSight, LANSpace, LANSpool, MAPNET, Matched, MCS, Media Mail, NetPort, NetSentry, OpenNET, PRO750, ProSolver, READY-LAN, Reference Point, RMX/80, SatisFAXtion, SnapIn 386, Storage Broker, SugarCube, The Computer Inside., TokenExpress, Visual Edge, and WYPIWYF.

MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

CG-011392

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION 1-1

1.1 CENTRAL PROCESSING UNIT (CPU) AND MEMORY 1-1
1.2 INTERRUPTS 1-2
1.3 ON-CHIP PERIPHERALS 1-2
1.3.1 Timers and the Event Processor Array (EPA) 1-2
1.3.2 Waveform Generator 1-3
1.3.3 Pulse Width Modulation Unit 1-3
1.3.4 A/D Converter 1-3
1.4 I/O PORTS 1-3
1.5 MODES OF OPERATION 1-3
1.6 SOFTWARE 1-4
1.7 NOTATION 1-4

CHAPTER 2
CPU AND MEMORY CONTROLLER 2-1

2.1 CPU OPERATION 2-1
2.1.1 CPU Control 2-1
2.1.2 Register Arithmetic Logic Unit (RALU) 2-1
2.2 MEMORY CONTROLLER 2-3

CHAPTER 3
MEMORY SPACE 3-1

3.1 RESERVED MEMORY 3-2
3.2 REGISTER FILE 3-2
3.2.1 Lower Register File 3-3
3.3 SPECIAL FUNCTION REGISTERS 3-4
3.4 INTERNAL ROM AND EPROM 3-4
3.5 EXTERNAL MEMORY AND ADDRESS/DATA BUS 3-5

CHAPTER 4
SOFTWARE OVERVIEW 4-1

4.1 OPERAND TYPES 4-1
BYTES 4-2
WORDS 4-2
SHORT INTEGERS 4-2
INTEGERS 4-2
BITS 4-2

DOUBLE WORDS 4-3
LONG INTEGERS 4-3
4.2 OPERAND ADDRESSING 4-3
Register-Direct References 4-3
Indirect References 4-4
Indirect with Auto-Increment References 4-4
Immediate References 4-5
Short-Indexed References 4-5
Long-Indexed References 4-6
ZERO_REG Addressing 4-6
Stack Pointer Register Addressing 4-6
Assembly Language Addressing Modes 4-7
4.3 WINDOWING 4-7
4.3.1 Areas that Can Be Windowed 4-8
4.3.2 Window Select Register (WSR) – Location 014H 4-8
4.3.3 Windowing and Addressing Modes 4-10
4.3.4 Windowing Examples 4-11
4.4 PROGRAM STATUS WORD (PSW) 4-12
PSW Flags 4-12
Interrupt Masks 4-14
4.5 INSTRUCTION SET 4-14
4.6 8XC196MC INSTRUCTION SET ADDITIONS 4-15
4.6.1 Table Indirect Jump (TIJMP) 4-22

CHAPTER 5
I/O PORT 2 5-1

5.1 CIRCUIT OPERATION AND REGISTERS 5-1
5.2 CONFIGURATION FOR STANDARD I/O 5-3
5.3 CONFIGURATION FOR SPECIAL FUNCTIONS 5-3

CHAPTER 6
EVENT PROCESSOR ARRAY 6-1

6.1 TIMER/COUNTER STRUCTURE 6-3
6.2 CAPTURE/COMPARE STRUCTURE 6-5
6.2.1 Capture/Compare Modules 6-5
6.2.2 Compare Modules 6-8

CHAPTER 7
WAVEFORM GENERATOR 7-1

7.1 WG Special Function Registers 7-1
7.1.1 Up/Down Counter (WG_COUNT) 7-3
7.1.2 Reload Register (WG_RELOAD) 7-3

7.1.3 Phase Compare Buffer Registers (WG_COMPx) 7-3
7.1.4 WG Control Register (WG_CON) 7-3
7.1.5 Output Control Buffer Register (WG_OUT) 7-3
7.2 WG MODES OF OPERATION 7-4
7.3 WAVEFORM GENERATING PROCESS 7-6
7.4 PROTECTION CIRCUIT 7-8
7.5 WG INTERRUPT 7-9
7.6 Applications Examples 7-10

CHAPTER 8
PULSE WIDTH MODULATION 8-1

8.1 PWM PERIOD REGISTER (PWM_PERIOD) 8-2
8.2 PWM PERIOD COUNT REGISTER (PWM_PER_CNT) 8-2
8.3 PWM0 AND PWM1 REGISTERS 8-2
8.4 WAVEFORM GENERATOR OUTPUT CONTROL REGISTER (WG_OUT) 8-2
8.5 PWM COUNTER 8-3

CHAPTER 9
ANALOG-TO-DIGITAL CONVERTER (PORT 0 AND 1) 9-1

9.1 A/D CONVERSION PROCESS 9-2
9.1.1 A/D Command Register (AD_COMMAND) 9-3
9.1.2 A/D Result Register (AD_RESULT) 9-4
9.2 A/D TIME REGISTER (AD_TIME) 9-4
9.2.1 Considerations for Setting SAM and CONV 9-6
9.3 A/D TEST REGISTER (AD_TEST) 9-7
9.4 A/D PORT STRUCTURE AND INTERFACE CIRCUITRY 9-7
9.4.1 A/D Port Structure 9-7
9.4.2 A/D External Interface Circuitry 9-8
9.5 THE A/D TRANSFER FUNCTION 9-10
9.6 A/D GLOSSARY OF TERMS 9-15

CHAPTER 10
MEMORY MAPPED I/O PORTS 3, 4, AND 5 10-1

10.1 PORTS 3 AND 4 10-1
10.1.1 Circuit Operation 10-2
10.1.2 Using Ports 3 and 4 10-2
10.2 PORT 5 10-3
10.2.1 Circuits and Registers for Port 5 10-4
10.2.2 Configuration for Standard I/O 10-6
10.2.3 Configuration for Special Functions 10-7
10.2.4 Port 5 Special Function Signals 10-7

CHAPTER 11
INTERRUPTS 11-1

11.1 INTERRUPT CONTROL 11-4
11.1.1 Interrupt Pending Register 11-4
11.1.2 Interrupt Mask Register 11-5
11.1.3 Global Interrupt Enable 11-5
11.2 SPECIAL INTERRUPTS 11-6
NMI 11-6
TRAP 11-6
Unimplemented Opcode 11-6
11.3 INTERRUPT PRIORITIES 11-6
11.4 INTERRUPT TIMING 11-8

CHAPTER 12
PERIPHERAL TRANSACTION SERVER (PTS) 12-1

12.1 PTS CONTROL 12-3
12.2 PTS TIMING 12-5
12.3 PTS MODES 12-5
12.3.1 Single Transfer Mode 12-6
12.3.2 Block Transfer Mode 12-7
12.3.3 A/D Scan Mode 12-8
12.3.4 PTS Serial I/O Mode (SIO) 12-10
12.3.4.1 PTS Asynchronous Serial I/O Mode (ASIO) 12-13
12.3.4.2 Example 1, Asynchronous Receive 12-15
12.3.4.3 Example 2, Asynchronous Transmit 12-16
12.3.4.4 PTS Synchronous Serial I/O Mode (SSIO) 12-18
12.3.4.5 Example 3, Synchronous Receive Mode 12-19
12.3.4.6 Example 4, Synchronous Transmit Mode 12-21
12.3.5 CPU Overhead of the PTS SIO Operation 12-22

CHAPTER 13
SPECIAL MODES OF OPERATION 13-1

13.1 IDLE MODE 13-1
13.2 POWERDOWN MODE 13-1
13.3 ONCE AND OTHER TEST MODES 13-3

CHAPTER 14
MINIMUM HARDWARE CONSIDERATIONS 14-1

14.1 POWER SUPPLY PINS 14-1

14.2 COMMON RETURN PINS 14-1
14.3 NOISE PROTECTION TIPS 14-2
14.4 OSCILLATORS AND INTERNAL TIMINGS 14-3
14.4.1 On-Chip Oscillator 14-3
14.4.2 Internal Timings 14-4
14.5 $\overline{\text{RESET}}$ AND RESET STATUS 14-4
14.5.1 $\overline{\text{RESET}}$ and Reset Circuits 14-5
14.5.2 Watchdog Timer (WATCHDOG) 14-7
14.5.3 RST Instruction 14-9
14.6 MINIMUM HARDWARE CONNECTIONS 14-10

CHAPTER 15
EXTERNAL MEMORY INTERFACING 15-1

15.1 BUS OPERATION 15-1
15.2 CHIP CONFIGURATION REGISTERS 15-3
15.3 BUS CONTROL 15-3
15.3.1 Wait States (Ready Control) 15-3
15.3.2 Bus Width (BUSWIDTH) and Memory Configurations 15-5
Memory Configuration Examples 15-6
15.3.3 Bus Control Modes 15-9
Standard Bus Control 15-9
Write Strobe Mode 15-10
Address Valid Strobe Mode 15-10
Address Valid with Write Strobe 15-11
15.4 AC TIMING EXPLANATIONS 15-10

CHAPTER 16
USING ROM AND EPROM PARTS 16-1

16.1 POWER-UP AND POWER-DOWN 16-1
Power-Up 16-1
Power-Down 16-2
16.2 PROGRAMMING THE 87C196MC 16-2
16.3 AUTO PROGRAMMING MODE 16-3
16.4 SLAVE PROGRAMMING MODE 16-5
16.4.1 Data Program Command 16-5
16.4.2 Word Dump Command 16-6
16.5 RUN-TIME PROGRAMMING 16-7
16.6 ROM/EPROM MEMORY PROTECTION OPTIONS 16-8
16.6.1 Authorized Access of Protected Memory 16-9
16.6.2 ROM Dump Mode 16-10
16.7 UPROMs 16-11
16.7.1 Programming UPROMs 16-11
16.8 Algorithms 16-12
16.8.1 Modified Quick Pulse Algorithm 16-12
16.8.2 Signature Word 16-12

APPENDIX A
PIN DESCRIPTIONS A-1

APPENDIX B
MCS®-96 INSTRUCTION SET B-1

Figures

1.1. 8XC196MC Block Diagram 1-1
2.1. Block Diagram of the Register File, RALU, Interrupt Controller and Memory Controller 2-2
3.1. CPU Special Function Registers and Stack Pointer 3-5
3.2. Internal Special Function Registers 3-6
4.1. WSR Window Values 4-11
4.2. The Program Status Word (PSW) Register (Upper Byte) 4-12
5.1. Circuit Schematic for Port 2 5-3
6.1. EPA Timer/Counters 6-2
6.2. A Single EPA Capture/Compare Module 6-3
6.3. EPA Block Diagram 6-4
6.4a. T1CONTROL Register 6-6
6.4b. T2CONTROL Register 6-7
6.5. Quadrature Timing Mode 6-7
6.6. Quadrature Mode Interface 6-8
6.7. CAPCOMPx_CON Register 6-9
6.8. Example CAPCOMPx_CON Operations 6-10
6.9. EPA Compare Control (COMPx_CON) 6-11
6.10. Example COMPx_CON Operations 6-12
7.1. Block Diagram of WG 7-2
7.2. WG Control Register 7-4
7.3. WG Output Control Register (WG_OUT) 7-5
7.4. Waveforms of Events 7-7
7.5. WG Protection Control Register 7-9
7.6. WG Protection Block Diagram 7-9
7.7. Peripheral Interrupt Mask Register 7-10
7.8. Peripheral Interrupt Pending Register 7-10
7.9. Phase1 Timing Diagram in Mode 1 with OP1=0 and OP0=0 7-12
7.10. Timing Diagram in Mode 3 with OP1=0 and OP0=0 7-13
7.11. Timing Diagram in Mode 2 with OP1=0 and OP0=0 7-14
8.1. PWM Block Diagram 8-1
8.2. WG Output Control Register (WG_OUT) 8-3
9.1. A/D Converter Block Diagram 9-1
9.2. AD_COMMAND Register 9-3
9.3. AD_RESULT Register 9-5
9.4. AD_TIME Register 9-6
9.5. Example Time Chart 9-6

9.6. AD_TEST Register ... 9-7
9.7. Port 0 and Port 1 Structure ... 9-8
9.8. Idealized A/D Sampling Circuitry ... 9-9
9.9. Suggested A/D Input Circuit ... 9-10
9.10. Ideal A/D Characteristic ... 9-12
9.11. Actual and Ideal A/D Characteristic ... 9-13
9.12. Terminal Based A/D Characteristic ... 9-14
10.1. Circuit Schematic for Ports 3 and 4 ... 10-2
10.2. Port 3 and 4 Logic Table ... 10-3
10.3. Circuit Schematic for Port 5 ... 10-5
10.4. Port 5 Logic Table ... 10-6
10.5. Register Settings for Standard I/O Port Operation ... 10-6
11.1. 8XC196MC Interrupt Structure Block Diagram ... 11-2
11.2a. Interrupt Pending and Mask Registers ... 11-4
11.2b. Interrupt Pending and Mask Registers ... 11-4
11.2c. Peripheral Interrupt Pending and Mask Registers ... 11-4
11.3. Interrupt Response Time ... 11-9
12.1. Flow Diagram for PTS and Normal Interrupts ... 12-2
12.2. The PTS_SEL and PTS_SRV Registers ... 12-3
12.3. PTS Latency ... 12-5
12.4. PTS Control Blocks (PTSCB) ... 12-6
12.5. PTS Control for Single Transfer ... 12-7
12.6. PTS Control for Block Transfer Mode ... 12-8
12.7. PTS_CONTROL for A/D Scan Mode ... 12-8
12.8. PTS SIO Control Block ... 12-11
12.9. PTSCON for the ASIO and SSIO Modes ... 12-12
12.10. PTSCON1 in ASIO Mode ... 12-14
12.11. Asynchronous Baud Rate Equation ... 12-14
12.12. Format of the DATA Register 8-Bit Receive Mode ... 12-14
12.13. Receive Data Byte and Description of Terms ... 12-15
12.14. Format of the DATA Register 8-bit Transmit Mode ... 12-17
12.15. Asynchronous Transmit Data Waveform (DATA = 55H) ... 12-17
12.16. PTSCON1 in SSIO Mode ... 12-18
12.17. Asynchronous Baud Rate Equation ... 12-19
12.18. Synchronous Receive ... 12-20
12.19. Synchronous Serial Receive Mode ... 12-21
13.1. Power Down and Power Up Sequences ... 13-2
14.1. Power and Return Connections ... 14-2
14.2. On-Chip Oscillator Circuitry ... 14-3
14.3. External Crystal Connections ... 14-4
14.4. Internal Clock Phase ... 14-5
14.5. Reset Sequence ... 14-7
14.6. Reset Pin ... 14-9
14.7. System Reset Circuit ... 14-9
14.8. 87C196MC Minimum Hardware Connections ... 14-10
15.1. Idealized Bus Timings ... 15-2
15.2. Chip Configuration Bytes ... 15-4

15.3. Bus Width Options 15-5
15.4. 8-Bit System with EPROM 15-6
15.5. 8-Bit System with EPROM and RAM 15-7
15.6. 16-Bit System with EPROM 15-7
15.7. 16-Bit System with Dynamic Bus Width 15-8
15.8. 16-Bit System with Single-Byte Writes to RAM 15-8
15.9. Standard Bus Control 15-9
15.10. Decoding $\overline{WRL}$ and $\overline{WRH}$ 15-9
15.11. Write Strobe Mode 15-10
15.12. Address Valid Strobe Mode 15-11
15.13. Address Valid with Write Strobe Mode 15-11
15.14. System Bus Timing 15-12
16.1. Programming Mode Pin Functions 16-3
16.2. Auto Programming Mode 16-6
16.3. Data Program Command 16-7
16.4. Word Dump Command 16-7
16.5. Run-Time Programming Algorithm Example 16-8
16.6. UPROM Special Function Register (USFR) 16-11

Tables

3.1. 8XC196MC Memory Map 3-1
3.2. Reserved Memory 3-3
4.1. 128-Byte Windows 4-8
4.2. 64-Byte Windows 4-9
4.3. 32-Byte Windows 4-10
4.4. Instruction Summary 4-16
4.4. Instruction Summary (Continued) 4-17
4.4. Instruction Summary (Continued) 4-18
4.5. Instruction Timing 4-19
4.5. Instruction Timing (Continued) 4-20
4.5. Instruction Timing (Continued) 4-21
4.6. TIJMP Operation 4-22
5.1. Port 2 Logic Table 5-4
5.2. Register Settings for Standard I/O Port Operation 5-5
7.1. WG Modes of Operation 7-6
11.1. Summary of Interrupts and Priorities 11-3
12.1. PTS Vector Table 12-4
12.2. A/D Scan Mode Table 12-10
14.1. Pin States during Reset, Idle, and Powerdown 14-6
14.2. SFR Reset Values 14-8
16.1. Programming Mode Pin Definitions 16-4
16.2. Programming Mode PMODE Values 16-4
16.3. Write Protection Options 16-9
16.4. Security Key Verification 16-10
16.5. Signature Word and Voltage Levels 16-12

Introduction 1

CHAPTER 1
INTRODUCTION

The 8XC196MC is a 16-bit microcontroller designed especially for the high-speed event control required for electric motor and invertor applications. The 8XC196MC has a 64 Kbyte address space. Figure 1.1 is a block diagram of the 8XC196MC. Its main components include a CPU, several types of memory, seven I/O ports and several on-chip peripheral devices. The peripherals include an A/D converter, an event processor array (EPA), two timers (Timer1 and Timer2), a 3-phase waveform generator (WG) and a pulse width modulation unit (PWM).

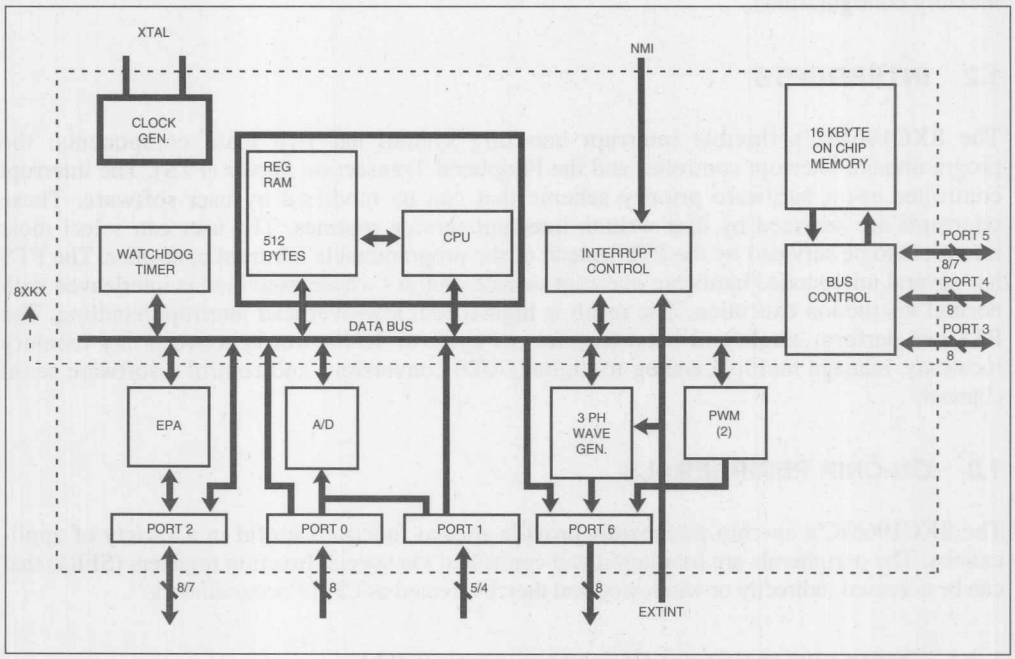


Figure 1.1. 8XC196MC Block Diagram

1.1 CENTRAL PROCESSING UNIT (CPU) AND MEMORY

The Central Processing Unit (CPU) consists of a Register Arithmetic-Logic Unit (RALU) and a 512-byte register file (the lower 24 bytes of the register file are reserved for SFR's and cannot be used as general purpose RAM). The RALU includes three temporary registers (two with their own shift logic), internal constants to speed calculations and a bit select register for generating single-bit masks. See Section 2.0 for more details.

The RALU does not use an accumulator. Instead it operates directly on the lower 256 bytes of the register file, which essentially provides 256 "accumulators." In addition, a windowing mechanism permits the RALU to also directly access the upper 256 bytes of the register file and the standard special function registers, which are located from 1F00H to 1FDFH. Direct access to these memory areas provides ease of programming, efficient code and fast execution. Other areas of memory are accessed via the memory controller, which includes a 4-byte prefetch queue to speed program execution and a bus controller. The bus controller manages access to three areas of memory: 16 Kbytes of on-chip ROM/EPROM, memory-mapped special function registers (SFRs) which are located from 1F40H to 1FFFH and external memory. The external memory is accessed via the address/data (ADDR/DATA) bus, which can operate in 8- or 16-bit modes. Several bus control signals can be selected to support a variety of external memory configurations.

1.2 INTERRUPTS

The 8XC196MC's flexible interrupt handling system has two main components: the programmable interrupt controller and the Peripheral Transaction Server (PTS). The interrupt controller has a hardware priority scheme that can be modified by user software. These interrupts are serviced by user-written interrupt service routines. The user can select most interrupts to be serviced by the PTS instead of the programmable interrupt controller. The PTS has several microcoded hardware interrupt service routines whose execution is interleaved with normal instruction execution. The result is high-speed, low-overhead interrupt handling. The PTS can perform single and burst transfers of bytes or 16-bit words between any memory locations, manage multiple analog-to-digital (A/D) conversions and control a software serial channel.

1.3 ON-CHIP PERIPHERALS

The 8XC196MC's on-chip peripherals provide special functions useful in a variety of applications. The peripherals are monitored and controlled via special function registers (SFRs) that can be accessed indirectly or windowed and thereby treated as CPU ''accumulators.''

1.3.1 Timers and the Event Processor Array (EPA)

The Event Processor Array (EPA) performs input and output functions associated with Timer1 and Timer2. In the input mode the EPA monitors an input pin for signal transitions and records the timer value when the event occurs. The "captured" event is thus tagged with its time. In the output mode the EPA waits until the timer matches a stored time value and then sets, clears or toggles an output pin. This is a "compare" event. Both capture and compare events initiate interrupts, which can be handled by a normal service routine or the PTS. The 8XC196MC has 4 capture/compare modules and 4 compare-only modules.

The two 16-bit timers can be clocked by the internal clock generator. Timer1 can also be clocked by external sources. An external ''quadrature clocking'' mode is available for monitoring speed and direction from a position encoder.

1.3.2 Waveform Generator

The Waveform Generator (WG) produces 3 pairs of complimentary PWM signals. This peripheral is optimized for controlling 3-phase induction AC motors. It can also control brushless DC motors and DC to AC inverters. A dead-time generator and phase inverter circuit provide non-overlapping on-times for each PWM output pair. Each signal is independently programmable.

1.3.3 Pulse Width Modulation Unit

The 8XC196MC has a PWM module that provides two PWM outputs. This module is in addition to the waveform generator. The duty cycle and the period of each output is programmable through a respective 8-bit period register. The module has an 8-bit counter, two 8-bit PWM compare registers and an 8-period register. The PWM output pins are controlled with bits in the output control register of the waveform generator.

1.3.4 A/D Converter

The 13-channel A/D converter can perform 10-bit conversions or faster 8-bit conversions. Automated A/D conversions and result storage are facilitated by the A/D scan mode of the PTS. The sample-and-hold times and the conversion times are programmable. The A/D can also act as a programmable comparator and issue an interrupt when the input crosses a threshold. Conversions can be performed on the analog ground and reference voltage, and the results can be used to calculate gain and zero offset errors. The zero offset compensation circuit is also programmable, enabling automatic offset adjustment.

1.4 I/O PORTS

The 8XC196MC has 7 I/O ports, labeled 0-6. Individual port pins are multiplexed to serve for standard I/O or to carry special signals. All ports are 8-bit except port 1 which is a 5-bit port.

Ports 0, 1 and 2 are controlled by SFRs that can be directly addressed by the RALU through a window in the register file. Ports 0 and 1 serve as input to the 13-channel A/D, and can also be read as digital inputs. Port 2 can be configured either as standard I/O ports or to serve special functions. Port 6 is the output port for the PWM and WG units.

Ports 3, 4 and 5 are memory mapped and cannot be windowed. These ports are accessed only via 16-bit addresses. Ports 3 and 4 also serve as the 16-bit external address/data bus. The Port 5 lines can be selected for standard I/O or to serve as system control pins.

1.5 MODES OF OPERATION

The 8XC196MC operates in several modes in addition to the normal execution mode. In Idle Mode the CPU stops executing while peripheral clocks continue active. Power consumption drops to about 40% of active mode power. In powerdown mode all internal clocks are frozen at

logic state zero and the oscillator is shut off. The register file, internal RAM and most peripherals hold their values if V_{CC} is maintained. Power is reduced to the device leakage and is in the μA range.

Test modes available to the user are the ONCE (ON-Circuit-Emulation) mode and several modes for programming and/or verifying the contents of on-chip EPROM/ROM.

The ONCE mode electrically removes the 8XC196MC from the rest of the system. A typical application of the ONCE mode is to test a circuit board while the device is soldered on the board.

The 8XC196MC can be placed in several EPROM programming modes. Three modes are available:

1. Auto programming mode described in Section 16.3, enables the device to program itself without a special EPROM programmer.
2. Slave mode described in Section 16.4, provides both a standard interface for programming by an EPROM programmer and a ROM dump mode for verifying the contents of ROM parts.
3. Run-Time mode described in Section 16.5, allows individual EPROM locations to be programmed at run-time under software control. Unlike the other modes, run-time programming is accomplished without entering the general EPROM programming mode.

1.6 SOFTWARE

The 8XC196MC instruction set is based on the 8096BH. It uses a variety of addressing modes and includes a full set of arithmetic and logical instructions for 8-bit and 16-bit data types. 32-bit data types are supported for the product of a 16-by-16 bit multiply, the dividend of a 32-by-16 divide and the operand in a shift operation. Combinations of 16-bit instructions easily implement the remaining 32-bit operations. Floating-point operations are supported by the floating-point library (FPAL-96), which implements a single precision subset of the proposed IEEE standard for floating-point operations.

Instructions were added to the 8096BH instruction set for the 8XC196MC. Four major ones are mentioned here. PUSHA pushes the Program Status Word (PSW), the interrupt mask registers (INT_MASK and INT_MASK1), and the Window Select Register (WSR) onto the stack. POPA pops the same registers. The TIJMP (table indirect jump) instruction reduces the overhead associated with identifying a multiplexed EPA interrupt source on a single interrupt request line. IDLPD places the 8XC196MC into the idle or powerdown mode.

1.7 NOTATION

Names of registers are in upper case. For example:

P5_PIN	The input register for Port 5

A lower case letter in a register name indicates that the name represents more than 1 register. For example:

Px_PIN (x=3,4)	The input register for Port 3 or 4
Px_REG	The output register for Ports 0-6

Bit locations in a 16-bit register are indexed by 0-15, where bit 0 is the least significant bit (LSB) and bit 15 is the most significant bit (MSB). A single bit in a register is denoted by the register name followed by a period and the bit number. For example:

P5_PIN.2	Bit 2 of P5_PIN
P5_PIN.x	Bit 0-7 of P5_PIN

The least significant *byte* of a 16-bit word is also denoted by LSB, and the most significant byte is denoted by MSB.

The following notation is used for decimal, hexadecimal and binary numbers. The decimal number 199 is represented as:

199	(decimal)
C7H	(hexadecimal)
1100 0111B	(binary)

CPU and Memory Controller

2

CPU and
Memory Controller

2

CHAPTER 2
CPU AND MEMORY CONTROLLER

Figure 2.1 is a block diagram of the 8XC196MC RALU, register file, memory controller and interrupt controller. The CPU consists of the RALU and the register file.

The CPU is 16 bits wide and is connected to the interrupt controller and the memory controller via the 16-bit CPU bus. An extension of the 16-bit CPU bus connects the CPU to the peripheral devices (Figure 1-1). In addition, an 8-bit CPU bus transfers instruction bytes from the memory controller to the instruction register in the RALU. The interrupt controller is described in Section 11.

2.1 CPU OPERATION

The major components of the 8XC196MC CPU (Figure 2.1) are the 512-byte register file and the register arithmetic logic unit (RALU). The register file has two parts: the lower register file (lower 256 bytes) and the upper register file (upper 256 bytes). As detailed in Section 3.2, the lower register file contains 24 bytes of standard (hardware) registers and 232 bytes of register RAM; all 256 bytes of the upper register file are register RAM.

The RALU does **not** use an accumulator. Instead, it operates directly on the 256-byte lower register file and thus has effectively 256 "accumulators." Additional RAM and SFRs outside the lower register file can also be directly addressed by the RALU through *windowing*, a mechanism that allows direct reference to these locations through a *window* in the lower register file. The register file, together with windowing, speed throughput and I/O operations (windowing is explained in Section 4.3). Memory outside the register file is accessed through the memory controller.

2.1.1 CPU Control

The CPU is controlled by the microcode engine (Figure 2.1), which instructs the RALU to perform operations with any byte, word or double word in (or windowed into) the 256-byte lower register file. Instructions for the CPU are taken from the 4-byte queue in the memory controller and temporarily stored in the instruction register. The microcode engine decodes the instructions and generates the correct sequence of events to have the RALU perform the desired function(s).

2.1.2 Register Arithmetic Logic Unit (RALU)

The 8XC196MC performs most calculations in the register arithmetic logic Unit (RALU). The RALU contains a 16-bit ALU and several registers in addition to the instruction register and microcode engine described above. Except for the 3-bit bit select register, the RALU register widths are 16 bits or 17 bits (16 plus a sign extension). Some of the registers can perform simple operations themselves, thus reducing the ALU's workload. Words enter the ALU through the A- and B-inputs; words can be complemented before entering the B-input.

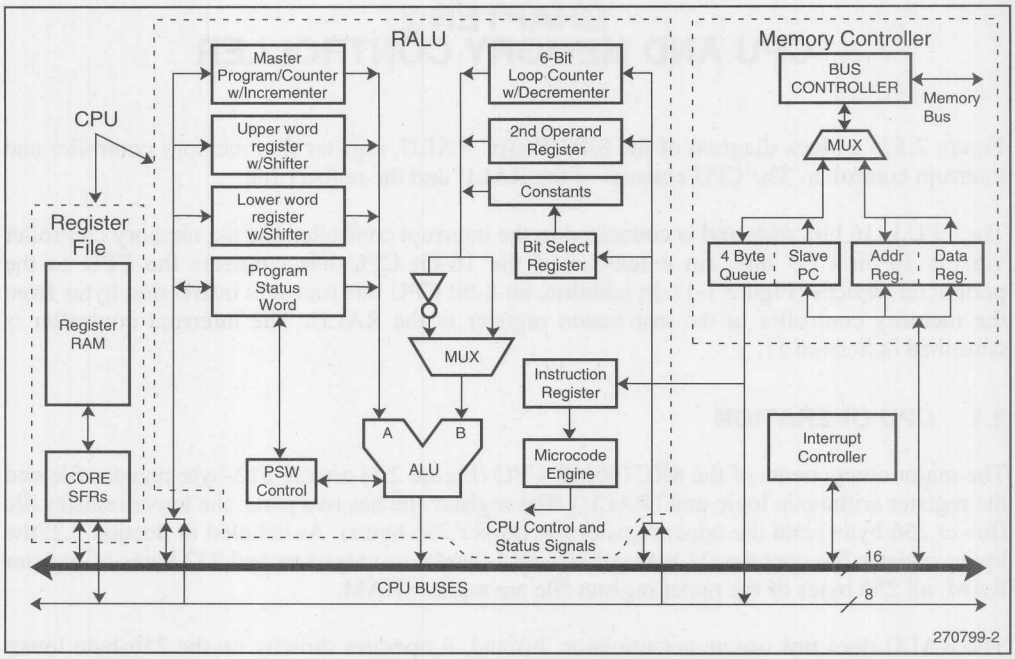


Figure 2.1. Block Diagram of the Register File, RALU, Interrupt Controller and Memory Controller

The Master Program Counter (PC), which holds the address of the next instruction, has its own incrementer. However, PC changes due to jumps, interrupts, calls or returns must be handled through the ALU. The program status word (described in Section 4.4) holds information concerning the state of the user's program.

The RALU employs three temporary registers: the upper and lower word registers and the second operand register. The upper and lower word registers are used together for the 32-bit instructions and as temporary registers for many instructions. They have their own shift logic and are used for operations that require logical shifts, including normalize, multiply and divide. Repetitive shifts are counted by the 6-bit loop counter. The second operand register is used in two-operand instructions to store the second operand. This includes the multiplier during multiplies and the divisor during divisions.

Several constants, such as ''0'', ''1'' and ''2'' are stored in the RALU to speed up calculations (e.g., to make a 2's complement number or perform an increment or decrement instruction). In addition, single bit masks for bit test instructions are generated in the constant register based on the 3-bit bit select register.

2.2 MEMORY CONTROLLER

The RALU communicates with the memory (except for the register file) through the memory controller (Figure 2.1). The memory controller includes the address and data registers, a 4-byte queue, a slave program counter (Slave PC) and a bus controller. The bus controller drives the internal ROM/EPROM bus, the Internal RAM bus and the external address/data bus, which are represented together in Figure 2.1 by the ''memory bus.'' Memory access requests to the bus controller come from either the RALU or the 4-byte queue, with queue accesses having priority. Requests from the queue are always for code fetches at the address in the Slave PC.

By having program fetches from memory locations referenced by the slave PC, the processor saves time as addresses seldom have to be sent from the master PC to the memory controller. If the address sequence changes because of a jump, interrupt, call or return, the slave PC is loaded with a new value from the PC, the queue is flushed and processing continues.

Execution speed is increased by using the pre-fetch queue as it usually keeps the next instruction byte available. This queue is transparent to the RALU and to the user.

NOTE

When using a logic analyzer to debug code, one must be aware of the pre-fetch queue. It is not possible to determine when an instruction will begin executing by simply watching when it is fetched, because the queue is filled in advance of instruction execution, and multiple fetches of the same opcode may occur.

Memory Space 3

CHAPTER 3
MEMORY SPACE

The addressable memory space on the 8XC196MC consists of 64 Kbytes (16-bits of address), most of which is available to the user for program or data memory. Each location holds one byte. A memory map is shown in Table 3.1. The areas labeled external memory are off-chip; the other areas are on-chip.

Table 3.1. 8XC196MC Memory Map

0FFFFH 06000H	External Memory		
05FFFH 02080H	Internal ROM/EPROM or External Memory		
0207FH 02000H	Reserved Memory (Internal ROM/EPROM or External Memory)		
01FFFH 01F00H	Internal Special Function Registers (SFRs)		
01EFFH 00200H	External Memory		
001FFH 00100H	Register RAM	Upper Register File (Address with indirect or indexed modes or through windows.)	Register File
000FFH 00018H	Register RAM	Lower Register File (Address with direct, indirect or indexed modes.)	
00017H 00000H	CPU SFRs		

The space 2000H-5FFFH can be occupied on-chip by either ROM or EPROM, and/or occupied off-chip by external memory. If on-chip ROM/EPROM is present, access to a location in this area can still be directed to external memory by the external access signal ($\overline{EA}$ = 0, Section 15). $\overline{EA}$ is latched upon reset.

Areas with special purposes are the register file (0000H-01FFH), the internal special function registers (1F00H-1FFFH), and reserved memory (2000H-207FH). All other areas, with the exception of individual "reserved" locations, can be used for either program or data storage or for memory mapped peripherals. Some individual memory locations outside reserved memory are marked "reserved." Such reserved locations should never be accessed. Reserved bit and byte locations in SFR space should always be written with a 0 (zero) (unless noted otherwise) to maintain compatibility with future devices in this family. Values read from these reserved

locations may vary from device to device. Registers and bits which are not labeled should be treated as reserved registers and bits.

Subsequent sections describe particular areas of on-chip memory, external memory and reserved memory. On-chip memory consists of: the register file, special function registers, internal RAM, ROM and EPROM

3.1 RESERVED MEMORY

Reserved memory (2000H-207FH) is an area set aside for special purposes. If the device has on-chip ROM/EPROM, reserved memory is located on-chip; otherwise, it may be in external memory. Table 3.2 shows the reserved memory in more detail. All addresses in this area marked ''reserved'' are reserved by Intel for use in testing or future products. All of this area, outside of SFR space, must be filled with the value 0FFH to ensure compatibility with future devices.

The interrupt vectors, described in Section 11, direct the user's program to interrupt service routines. The peripheral transaction server (PTS) vectors (Section 12) point to control blocks for the PTS units. The PTS services hardware interrupts with minimum CPU overhead. The Chip Configuration Bytes, CCB0 and CCB1, identifies the type of environment the 8XC196MC is operating in. The CCBs control features such as ROM/EPROM protection, the powerdown mode, and the watchdog timer. The CCBs are described in Section 15.2.

The security key (2020H-202FH) protects the 87C196MC against unauthorized reading and writing of ROM/EPROM. If the 87C196MC attempts to operate in a mode requiring authorization, the 16-byte programmable security key is compared byte by byte to a set of bytes supplied externally by the user. If the externally supplied data does not match the security key, the requested execution mode is not entered. The security key is described further in Section 16.6. The programming voltages and the signature word are described in Section 16.8.2.

3.2 REGISTER FILE

The 512 locations 000H-1FFH are the register file, composed of the 256-byte lower register file and the 256-byte upper register file. The register file resides on-chip in the CPU. Register RAM consists of locations 018H-1FFH, which are available for general data storage excepting possibly the stack pointer (SP, locations (018H and 019H) as noted in Section 3.2.1). Locations 00H-017H are the CPU special function registers (SFRs), discussed in Section 3.3. (An additional 256 bytes of SFR space to control the peripherals are at locations 1F00H-1FFFH).

Code cannot be executed from the register file. If an attempt is made to execute instructions from locations 000H-1FFH, the instructions will be fetched from external memory. This section of external memory is reserved for use by Intel development tools.

The register file, as well as the status of the majority of the chip, is kept intact while the chip is in powerdown mode as discussed in Section 13.2.

Table 3.2. Reserved Memory

0207FH 02074H	Reserved*
02073H 02072H	Programming Voltages
02071H 02070H	Signature Word
0206FH 0205EH	Reserved*
0205DH 02040H	Peripheral Transaction Server (PTS) Vectors
0203FH 02030H	Interrupt Vectors (Upper)
0202FH 02020H	Security Key
0201FH 0201EH 0201DH 0201CH	20H** Reserved* 20H** Reserved*
0201BH 0201AH 02019H 02018H	20H** CCB1 (Chip Configuration Byte 1) 20H** CCB0 (Chip Configuration Byte 0)
02017H 02014H	Reserved*
02013H 02000H	Interrupt Vectors (Lower)

*These locations should be filled with all ones (0FFH).

**These locations should be filled with 20H.

3.2.1 Lower Register File

The 256-byte lower register file (00H-FFH) is unique; the RALU can operate directly on any of these locations. The register RAM from locations 018H to 0FFH contains 232 bytes of RAM which can be accessed as bytes (8 bits), words (16 bits) or double words (32 bits). As each of these locations can be accessed directly by the RALU, there are essentially 232 general purpose ''accumulators.'' This architecture reduces accumulator bottleneck and speeds throughput.

Locations 18H and 19H contain the stack pointer (SP). These are not special function registers and may be used as standard RAM when stack operations are not being performed. Since SP is in the lower register file, the RALU can easily operate on it. If used as a stack pointer, SP must be initialized by the user program and can point to anywhere in the 64K memory space (internal or external). Operations on the stack cause it to build down, so SP should be initialized to 2 bytes above the highest stack location and must contain a word (even) address.

3.3 SPECIAL FUNCTION REGISTERS

Special Function Registers (SFRs) are registers for I/O control and other special functions. All of the peripheral devices on the 8XC196MC are controlled through these registers. The CPU SFRs, locations 00H-17H in the lower register file, are listed in Figure 3.1. They include locations with special contents, such as the Window Select Register (WSR), and I/O registers, such as the Watchdog Timer (WATCHDOG). The Internal SFRs (locations 1F00H-1FFFH listed in Figure 3.2) are for I/O control and are physically located in the on-chip peripherals. Locations in 1F00H-1FFFH not listed in Figure 3.2 should be regarded as ''reserved.'' SFRs can be addressed as bytes or words unless otherwise specified. Unused SFRs should be filled with 00H (all zeros). The individual SFRs are described in the relevant sections of this guide.

A programmer must be cautious when using an SFR as a source of operations or as a base or index register for indirect or indexed operations. As external events can change SFRs, and some SFRs are cleared when read, results may be different than expected. The potential for an SFR to change value must be recognized when using these registers. This is particularly important when high-level languages are used, as they do not always make allowances for SFR-type registers.

NOTE

SFR locations 1FE0H-1FFFH must be addressed using 16-bit addresses only (indexed addressing). These addresses cannot be windowed. (Section 4.3 explains windowing.) If an SFR in this area is read through the window, it will **appear** to contain FFH (all ones). Attempts to write to these locations through the window will have **no** effect on these SFRs. In addition, locations marked with a * must only be accessed as words.

3.4 INTERNAL ROM AND EPROM

Internal ROM or EPROM is an optional component of the 8XC196MC. If present, internal ROM/EPROM occupies memory locations which include the reserved memory area (2000H-207FH) and space open to the programmer (2080H-5FFFH). Upon reset, instructions are fetched beginning at location 2080H. (Reset is discussed in Section 14.5.) A reference to memory area 2000H-5FFFH is directed to internal ROM/EPROM if signal $\overline{EA}$ (external access) is high. Otherwise, the address is sent to the external memory bus. $\overline{EA}$ must be held low on devices without internal ROM or EPROM. The value of $\overline{EA}$ is latched upon reset.

The default value in ROM/EPROM locations is FFH (all ones). Section 16 discusses the use of ROM and EPROM.

19H	SP(HI)	0CH	reserved	
18H	SP(LO)	0BH	reserved	
17H	reserved	0AH	WATCHDOG	
16H	reserved	09H	INT_PEND	
15H	reserved	08H	INT_MASK	
14H	WSR	07H	PTSSRV(HI)	
13H	INT_MASK1	06H	PTSSRV(LO)	
12H	INT_PEND1	05H	PTSSEL(HI)	
11H	reserved	04H	PTSSEL(LO)	
10H	reserved	03H	reserved	
0FH	reserved	02H	reserved	
0EH	reserved	01H	ZERO_REG(HI)	
0DH	reserved	00H	ZERO_REG(LO)	

Figure 3.1. CPU Special Function Registers and Stack Pointer

3.5 EXTERNAL MEMORY AND ADDRESS/DATA BUS

A variety of external memory configurations and their control via the address/data bus are detailed in Section 15. The basic address/data bus timing also applies to two sections of on-chip memory: Internal RAM and ROM/EPROM. As many 8XC196MC operations involve this timing and the associated signals, reading Section 15.1 on the basic timing may be helpful for understanding other topics in this guide.

External memory is addressed at locations 200H-1EFFH and 6000H-FFFFH. Further, access to the area 2000H-5FFFH, possibly occupied by on-chip ROM/EPROM, can be redirected to external memory by setting the external access signal $\overline{EA}$ = 0. $\overline{EA}$ is latched at reset, i.e., the 8XC196MC directs memory accesses according to the value of $\overline{EA}$ at reset, and does not recognize a new value of $\overline{EA}$ until another reset is asserted. Reset is discussed in Section 14.5. Unused program memory should be set to FFH.

The address/data bus operates in several modes. The standard 16-bit bus mode multiplexes 16-bit addresses and 16-bit data. A second mode (the 8-bit mode) uses a 16-bit address and 8-bit data. A third mode dynamically switches between these 8-bit and 16-bit modes.

The 8XC196MC can insert 0 to 3 wait states without external logic to accommodate slow external memory. This is controlled by the IRC0-2 bits in CCB0 and CCB1 (see Section 15.2). Wait states can be extended beyond this by setting the appropriate ready control mode.

1FFFH P4_PIN	1FC0H WG_OUT	1F68H - 1F71H reserved
1FFEH P3_PIN	1FBFH reserved	1F66H COMP3_TIME*
1FFDH P4_REG	1FBEH PI_PEND	1F65H reserved
1FFCH P3_REG	1FBDH RESERVED	1F64H COMP3_CON
1FF8H- 1FFBH reserved	1FBCH PI_MASK	1F62H COMP2_TIME*
1FF7H P5_PIN	1FB7H - 1FBBH reserved	1F61H reserved
1FF6H USFR	1FB6H PWM_PER_CNT	1F60H COMP2_CON
1FF5H P5_REG	1FB5H reserved	1F5EH COMP1_TIME*
1FF4H reserved	1FB4H PWM_PERIOD	1F5DH reserved
1FF3H P5_DIR	1FB3H reserved	1F5CH COMP1_CON
1FF2H reserved	1FB2H PWM1	1F5AH COMP0_TIME*
1FF1H P5_MODE	1FB1H reserved	1F59H reserved
1FD7H- 1FF0H reserved	1FB0H PWM0	1F58H COMP0_CON
1FD6H P2_PIN	1FAFH AD_TIME	1F50H- 1F57H reserved
1FD5H reserved	1FADH- 1FAEH AD_TEST	1F4EH CAPCOMP3_TIME*
	1FADH reserved	
1FD4H P2_REG	1FACH AD_COMMAND	1F4DH reserved
1FD3H reserved	1FAAH AD_RESULT	1F4CH CAPCOMP3_CON
1FD2H P2_DIR	1FA9H P1_PIN	1F4AH CAPCOMP2_TIME*
1FD1H reserved	1FA8H P0_PIN	1F49H reserved
1FD0H P2_MODE	1fa7h-1F80H reserved	1F48H CAPCOMP2_CON
1FCFH reserved	1F7EH TIMER2*	1F46H CAPCOMP1_TIME*
1FCEH WG_PROTECT	1F7DH reserved	1F45H reserved
1FCCH WG_CON	1F7CH T2CONTROL	1F44H CAPCOMP1_CON
1FCAH WG_COUNT	1F7AH TIMER1*	1F42H CAPCOMP0_TIME*
1FC8H WG_RELOAD	1F79H reserved	1F41H reserved
1FC6H WG_COMP3	1F78H T1CONTROL	1F40H CAPCOMP0_CON
1FC4H WG_COMP2	1F74H- 1F77H reserved	1F00H-1F3FH reserved
1FC2H WG_COMP1	1F72H T1RELOAD	

*These registers can only be addressed as word registers.

NOTE: Reserved SFR's must be filled with 00H.

Figure 3.2. Internal Special Function Registers

Software Overview 4

CHAPTER 4
SOFTWARE OVERVIEW

This section provides information on writing 8XC196MC programs, including operand types, operand addressing, the program status word and a summary of instructions. The few 8XC196MC instructions that differ from those for the 8096BH and the 80C196KB are described in Section 4.6. As discussed in Section 2.1.2, the ALU can operate directly on the lower register file. This direct access can be extended to some other areas of memory by a windowing technique described in Section 4.3.

Additional software information can be found in the following documents:

MCS-96 Macro Assembler User's Guide	Order Number 122351 (DOS Systems)
MCS-96 Utilities User's Guide	Order Number 122356 (DOS Systems)
PL/M-96 User's Guide	Order Number 122361 (DOS Systems)
iC-96 User's Guide	Order Number 481194 (DOS Systems)

Throughout this chapter short sections of code are used to illustrate the operation of the device. It is assumed that the following set of temporary registers has been declared in lower register file RAM:

AX, BX, CX, DX are 16-bit registers.
AL is the low byte of AX.
AH is the high byte of AX.
BL is the low byte of BX.
CL is the low byte of CX.
DL is the low byte of DX.

These are the same as the names for the general data registers used in the 8086/80186. It is important to note that in the 8XC196MC these are not dedicated registers but merely the symbolic names assigned by the programmer to an 8-byte region **within the on-chip lower register file**.

4.1 OPERAND TYPES

The MCS-96 architecture supports a variety of data types useful in control applications. In the following discussion the names adopted by the PL/M-96 and iC-96 programming languages are used where appropriate. To avoid confusion, the name of an operand type is capitalized. A ''BYTE'' is an unsigned 8-bit variable; a ''byte'' is an 8-bit unit of data of any type. Similarly, a ''WORD'' is an unsigned 16-bit variable, while a ''word'' is a 16-bit unit of data of any type.

BYTES

BYTES are unsigned 8-bit variables with values from 0 to 255. Arithmetic and relational operators can be applied to BYTE operands, but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bit-wise. Bits within BYTES are labeled from 0 to 7, with 0 being the least significant bit. There are no alignment restrictions for BYTES, so they may be placed anywhere in the MCS-96 address space.

WORDS

WORDS are unsigned 16-bit variables with values from 0 to 65535. Arithmetics and relational operators can be applied to WORD operands, but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bit-wise. Bits within words are labeled from 0 to 15, with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

SHORT INTEGERS

SHORT INTEGERS are 8-bit signed variables with values from −128 to +127. Negative numbers are represented in 2's complement form. Arithmetic operations that generate results outside of the range of a SHORT INTEGER set the overflow indicators in the program status word. The numeric result returned is the same as for the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT INTEGERS; they may be placed anywhere in the MCS-96 address space.

INTEGERS

INTEGERS are 16-bit signed variables with values from −32,768 to +32,767. Arithmetic operations that generate results outside of the range of an INTEGER set the overflow indicators in the program status word. The numeric result returned is the same as for the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

BITS

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8XC196MC provides for the direct testing of any bit in the lower register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS; it does not support the direct addressing of bits as does the MCS-51 architecture.

DOUBLE WORDS

DOUBLE WORDS are unsigned 32-bit variables with values from 0 to 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for the operand in a shift, the dividend of a 32-by-16 divide, and the product of a 16-by-16 multiply. For these operations a DOUBLE WORD variable must reside in the lower register file and be aligned at an address which is evenly divisible by 4. A DOUBLE WORD operand is addressed by the address of its least significant byte. DOUBLE WORD operations that are not directly supported can be easily implemented with two WORD operations. For consistency with Intel provided software, the user should adopt the conventions for addressing DOUBLE WORD operands discussed in the latest 16-Bit Embedded Controller handbook.

LONG INTEGERS

LONG INTEGERS are 32-bit signed variables with values from −2,147,483,648 to +2,147,483,647. The MCS-96 architecture provides direct support for this data type only for the operand in a shift, the dividend of a 32-by-16 divide, and the product of a 16-by-16 multiply.

LONG INTEGERS can also be normalized (shifted left until the most significant bit is 1) with the second operand recording the shift count. For these operations a LONG INTEGER variable must reside in the lower register file and be aligned at an address which is evenly divisible by 4. A LONG INTEGER is addressed by the address of its least significant byte.

LONG INTEGER operations that are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands discussed in the latest 16-Bit Embedded Controller handbook.

4.2 OPERAND ADDRESSING

Operands are accessed within the 64K address space with six basic addressing modes. Some of the details of how these addressing modes work are hidden in the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section describes the addressing modes as they are handled by the hardware. The six basic addressing modes are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed and long-indexed. Two other useful addressing modes that employ the zero register (ZERO_REG) and the stack pointer (SP) are also described. The addressing modes are illustrated using instructions summarized in Table 4.4 and described in the *MCS-96 Macro Assembler User's Guide*.

Register-Direct References

The register-direct mode is used to directly access a register in the 256-byte on-chip lower register file. With the use of windowing (Section 4.3), this mode can also be used to directly

access the additional SFRs (1F00H-1FDFH) or the 256 bytes of the upper register file through a window in the 256-byte lower register file.

The register is selected by an 8-bit field within the instruction, and the register address must conform to the operand type's alignment rules. Depending on the instruction, up to three registers can take part in the calculation.

Examples:

```
ADD   AX,BX,CX          ; AX:=BX+CX
MUL   AX,BX             ; AX:=AX*BX
INCB  CL                ; CL:=CL+1
```

Indirect References

The indirect mode accesses a WORD in the lower register file containing the 16-bit operand address. The operand address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the 64K address space, including the lower register file. The register containing the indirect address is selected by an 8-bit field within the instruction. An instruction may contain only one indirect reference, and the remaining operands (if any) must be register-direct references.

Examples:

```
LD    BX,[AX]           ; BX:=mem_word(AX)
```

In this example, assume that before execution:

```
contents of AX      = 2FC2H
contents of 2FC2H   = 3F26H
```

Then after execution:

```
contents of BX      = 3F26H
```

```
ADDB  AL,BL,[CX]        ; AL:=BL+mem_byte(CX)
POP   [AX]              ; mem_word(AX):=mem_word(SP)
                        ; SP:=SP+2
```

Indirect with Auto-Increment References

This addressing mode is the same as the indirect mode except that the variable that contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT INTEGERS, the indirect address variable is incremented by one; if the instruction operates on WORDS or INTEGERS the indirect address will be incremented by two.

Examples:

```
LD      BX,[AX]+            ; BX:=mem_word(AX)
                            ; AX:=AX+2
```

In this example, if conditions were the same as those in the load (LD) example for the indirect reference mode, the results would be the same, except that after execution the contents of AX would be 2FC4H.

```
ADDB AL,BL,[CX]+            ; AL:=BL+mem_byte(CX)
                            ; CX:=CX+1

PUSH [AX]+                  ; SP:=SP - 2
                            ; mem_word(SP):=mem_word(AX)
                            ; AX:=AX+2
```

Immediate References

In this addressing mode an operand itself is in a field in the instruction. For operations on BYTE or SHORT INTEGER operands this field is 8-bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction may contain only one immediate reference; the remaining operand(s) must be register-direct references.

Examples:

```
ADD     AX, #340            ; AX:=AX+340
                            ; (decimal)
PUSH    #1234H              ; SP:=SP-2
                            ; mem_word(SP):=1234 (hex)
DIVB    AX, #10             ; AL:=AX/10 (decimal)
                            ; AH:=AX mod 10
```

Short-Indexed References

In this addressing mode an 8-bit field in the instruction selects a WORD variable in the lower register file that contains an address. This WORD variable is enclosed in square brackets. A second 8-bit field in the instruction stream is sign extended and summed with the WORD variable to form an operand address.

Since the 8-bit field is sign extended, the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction may contain only one short-indexed reference; the remaining operand(s) must be register-direct references.

Examples:

```
LD      AX,4[BX]            ; AX:=mem_word(BX+4)
```

In this example, assume that before execution:

contents of BL are 52H
contents of BH are A1H

The operand address is then A152H + 04H = A156H

```
MULB AX,BL,4[CX]          ; AX:=BL*mem_byte(CX+4)
```

Long-Indexed References

This addressing mode is like the short-indexed mode except that a **16-bit** field is taken from the instruction and added to the WORD variable to form the operand. No sign extension is necessary. An instruction may contain only one long-indexed reference, and the remaining operand(s) must be register-direct references.

Examples:

```
AND   AX,BX,TABLE[CX]     ; AX:=BX and mem_word(TABLE+CX)
ST    AX,TABLE[BX]        ; mem_word(TABLE+BX):=AX
ADDB  AL,BL,LOOKUP[CX]    ; AL:=BL+mem_byte(LOOKUP+CX)
```

ZERO_REG Addressing

The first two bytes in the lower register file (ZERO_REG) are fixed at zero by the 8XC196MC hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly. Since this mode uses indexed addressing, accesses are slower than register-direct accesses.

Examples:

```
ADD   AX,1234[0]          ; AX:=AX+mem_word(1234)
POP   5678[0]             ; mem_word(5678):=mem_word(SP)
                          ; SP:=SP+2
```

Stack Pointer Register Addressing

The system stack pointer (SP) in the 8XC196MC resides in locations 18H and 19H of the lower register file and is addressed as register 18H. In addition to providing for convenient manipulation of the stack pointer, this also facilitates accessing operands in the stack. The top of the stack, for example, can be accessed by using SP as the WORD variable in an indirect

reference. In a similar fashion, SP can be used in the short-indexed mode to access data within the stack.

Examples:

```
PUSH    [SP]            ; Duplicate top-of-stack
                        ; SP:=SP-2

LD      AX,2[SP]        ; AX:=Next-to-top
```

Assembly Language Addressing Modes

The MCS-96 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembler chooses between register-direct addressing and long-indexed addressing with ZERO_REG depending on the memory location of the operand. The user can simply refer to the operand by its symbolic name. If the operand is in the lower register file, a register-direct reference is used; if the operand is elsewhere in memory, a long-indexed reference is generated.

Indexed Addressing. The assembler chooses between short and long indexed references depending on the value of the index expression. If the value can be expressed in 8 bits, short indexing is used; if it cannot be expressed in 8 bits, long indexing is used.

These features of the assembly language simplify the programming task and should be used wherever possible.

4.3 WINDOWING

Windowing is a technique for expanding the amount of memory that can be accessed with register-direct addressing. This is desirable because some of the 8XC196MC instructions allow only 8-bit (register-direct) addressing (Section 4.2), which provides for ease of programming and efficient code. Windowing allows register-direct addressing of certain locations above FFH.

When windowing is enabled, directly addressing a location in a specified block of the lower register file accesses the contents of a location in a specified block of higher memory. This block in the lower register file is called the *window*. The block in higher memory is then seen through the window, or, we say that the block in higher memory is being *windowed*. The window and the block to be windowed are specified by the window select register (WSR) described in Section 4.3.2.

The window is set up as a block in the top of the lower register file. This can be chosen to be a 128-byte block (80H-FFH), a 64-byte block (C0H-FFH) or a 32-byte block (E0H-FFH).

4.3.1 Areas that Can Be Windowed

The locations that can be windowed are 128-byte, 64-byte or 32-byte blocks chosen from two areas. The first area is the register file (000H-1FFH). (Of course, the portion 00H-FFH is already in the lower register file and would not normally be windowed.) The second area contains the internal special function registers (1F00H-1FDFH).

Tables 4.1, 4.2 and 4.3 show the base (lowest) addresses of the 128-byte, 64-byte and 32-byte blocks of memory that can be windowed with the 8XC196MC. Also included in the figures are the base addresses of additional blocks that can be windowed by future family products. These blocks cannot be windowed with the 8XC196MC. The WSR contents are explained in the following section.

CAUTION

The top 32 bytes of SFRs (1FE0H-1FFFH) cannot be windowed. Reading these locations through a window causes their contents to appear as FFH (all ones); writing to these locations through the window has no effect. Thus, the internal SFR area that can be usefully windowed is 1F00H-1FDFH (224 bytes). Attempting to window an area outside the two supported areas (000H-1FFH, 1F00H-1FDFH) produces invalid results. A read of an unsupported location produces FFH, while a write has no effect.

Table 4.1. 128-Byte Windows

Address to Remap	WSR Contents
FFFFH 2000H	Windowing not possible
1F80H* 1F00H	1FH 1EH
1E80H 0200H	Windowing not possible
1080H 0100H 0080H 0000H	13H 12H 11H 10H

Window in lower register file: 80H-FFH

*Windowing of 1FE0H-1FFFH is not functional.

4.3.2 Window Select Register (WSR) – Location 014H

Addressing a particular memory location through a window requires three pieces of information: (i) the size of the window, (ii) the block to be windowed and (iii) the address of the location relative to the base address of the block. As shown in Figure 4.1, the window select register (WSR) specifies the size of the window and the block to be windowed. The direct address specifies the address of the particular location relative to the block base address.

Table 4.2. 64-Byte Windows

Address to Remap	WSR Contents
FFFFH 2000H	Windowing not possible
1FC0H* 1F80H 1F40H 1F00H	3FH* 3EH 3DH 3CH
1EC0H 0200H	Windowing not possible
01C0H 0180H 0140H 0100H 00C0H 0080H 0040H 0000H	27H 26H 25H 24H 23H 22H 21H 20H

Window in lower register file: C0H-FFH

*Windowing of 1FE0H-1FFFH is not functional.

Notice that 11 bits (0, 1, 2, ... 9, 10) specify the block and the address relative to the base address. These bits are split between the WSR and the direct address. The direct address uses just enough bits for the size of the block: 5 bits for 32 locations, 6 for 64 locations or 7 for 128 locations. The remaining bits in the WSR are just enough to enumerate the blocks that can be windowed: 6 bits for the 64 blocks with 32 bytes, 5 bits for the 32 blocks with 64 bytes or 4 bits for the 16 blocks with 128 bytes. Tables 4.1, 4.2 and 4.3 show the WSR contents for the different possibilities. The next section gives examples of windowing.

If WSR bits 4-6 are all zeros, WSR bits 0-3 must also be zeros. Other selections of bits 0-3 are reserved. Clearing the WSR deactivates windowing and restores access to the lower register file locations used for the window.

Once the WSR is set up for a window, the windowed (upper memory block) locations can be accessed through the window and also by the usual 16-bit addressing. The register file locations that are covered by the window are always accessible by indirect or indexed operations. They can be made directly accessible again by clearing the WSR or, in the case of a 64- or 128-byte window, by selecting a smaller window that does not include a location to which direct access is desired.

Table 4.3. 32-Byte Windows

Address to Remap	WSR Contents
FFFFH 2000H	Windowing not possible
1FE0H* 1FC0H 1FA0H 1F80H 1F60H 1F40H 1F20H 1F00H	7FH* 7EH 7DH 7CH 7BH 7AH 79H 78H
1EE0H 0200H	Windowing not possible
01E0H 01C0H 01A0H 0180H 0160H 0140H 0120H 0100H 00E0H 00C0H 00A0H 0080H 0060H 0040H 0020H 0000H	4FH 4EH 4DH 4CH 4BH 4AH 49H 48H 47H 46H 45H 44H 43H 42H 41H 40H

Window in lower register file: E0H-FFH

*Windowing of 1FE0H-1FFFH is not functional.

4.3.3 Windowing and Addressing Modes

Once the WSR is set up for a window, windowing is operative for any register-direct access that refers to a location in the window. Windowing has no effect on a location referenced by indirect, indexed or zero-register addressing.

	WSR								DIRECT ADDRESS							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
128:	0*	0*	0*	1*	10	9	8	7	1*	6	5	4	3	2	1	0
64:	0*	0*	1*	10	9	8	7	6	1*	1*	5	4	3	2	1	0
32:	0*	1*	10	9	8	7	6	5	1*	1*	1*	4	3	2	1	0

0, 1, 2, 3...9, 10 represent remapped address

*This bit is always as indicated, not the value of "bit 0" or "bit 1"

Figure 4.1. WSR Window Values

4.3.4 Windowing Examples

Example 1. A 64-byte window in the upper SFR space (1F40H-1F7FH) is to be windowed. The location to be accessed is 1F6CH.

A 64-byte window requires WSR.6 = 0 and WSR.5 = 1. From Table 4.2, the base address 1F40H corresponds to the WSR contents 0011 1101B. Location 1F6CH is 44 locations (1F6CH-1F40H = 2CH = 44) above the base address. The direct address is then 2CH (0010 1100B) + C0H (1100 0000B) = ECH (1110 1100B) including the required ones in Figure 4.1.

Example 2. Location 14BH is to be accessed through a 32-byte window.

Location 14BH = 331 is in the upper register file. From Table 4.3, the base address is 140H and corresponds to the WSR contents 0100 1010B. The direct address is then 14BH − 140H = 0BH (0000 1011B) + E0H (1110 000B) = EBH (1110 1011B).

Accesses to the lower register file addresses E0H-FFH are now redirected to memory addresses 140H-15FH, respectively. 16-bit addressing to memory locations 140H-15FH will read/write to 140H-15FH as before.

To illustrate the effects of different addressing modes for this window, consider the following section of code.

```
1. LDB   WSR, #4AH      ;set up window, 32 bytes, 0140H base
2. LD    20H, 0E6H      ;value at 146H,147H to 20H,21H
3. LD    20H, 146H[0]   ;value at 146H,147H to 20H,21H
4. LD    0E0H, 0E6H     ;value at 146H,147H to 140H,141H
5. LDB   0E2H, [0E6H]   ;value at address contained in E6H to 142H
6. LDB   0E3H, 0E6H[0]  ;value at E6H to 143H
7. CLRB  WSR            ;cancel windowing
```

line 2: E6H is in the window; 20H is not.
line 3: 146H,147H can be accessed without windowing.
line 4: E6H and E0H are both in the window.
line 5: Windowing does not apply to indirect addressing of E6H.
line 6: Windowing not effective with zero indexed addressing.

Example 3. Location 1FE6H is to be accessed through a 128-byte window.

Location 1FE6H is in the top block in Table 4.1. However, it is above the top location (1FDFH) that can be usefully windowed. Although the window could be set up, reading this location through the window would yield FFH, and writing to it would not change the contents.

4.4 PROGRAM STATUS WORD (PSW)

The program status word (PSW) consists of two bytes. The upper byte is a collection of Boolean flags which retain information concerning the state of the user's program. Table 4.4 in Section 4.6 shows which flags can be affected by each instruction. The lower byte is the interrupt mask (INT_MASK), discussed later in this section and in Section 11.1.2. The PSW can be saved in the system stack with a single operation (PUSHF/PUSHA) and restored in a like manner (POPF/POPA). Only the interrupt byte INT_MASK of the PSW can be accessed directly. There is no SFR for the PSW flags. Figure 4.2 shows the flags of the PSW with their bit locations.

Bit	15	14	13	12	11	10	9	8
Flag	Z	N	V	VT	C	PSE	I	ST

Figure 4.2. The Program Status Word (PSW) Register (Upper Byte)

PSW Flags

Z: The Zero flag is set when an operation generates a result equal to zero. The Z flag is never set by the add with carry (ADDC/ADDCB) or subtract with carry (SUBC/SUBCB) operations, but it is cleared if the result is non-zero. These two instructions are normally used in conjunction with ADD/ADDB and SUB/SUBB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N: The Negative flag is set when an operation generates a negative result. Note that the N flag will be in the algebraically correct state even if overflow occurs. For shift operations, including the normalize operation and all three forms (SHL, SHR, SHRA) of byte, word and double word shifts, the N flag is set to the same value as the most significant bit of the result. This is true even if the shift count is zero.

V: The oVerflow flag is set when an operation generates a result that is outside the range for the destination data type. For SHL, SHLB, SHLL instructions, the V flag is set if the most significant bit of the operand changes at any time during the shift. For divide operations, the following conditions are used to determine if the V flag is set:

For the operation	V is set if quotient is:
Unsigned Byte Divide	> 255 (FFH)
Unsigned Word Divide	> 65535 (FFFFH)
Signed Byte Divide	< −128 (81H) or > +127 (7FH)
Signed Word Divide	< −32768 (8001H) or > 32767 (7FFFH)

VT: The oVerflow Trap flag is set when the V flag is set, but it is only cleared by the CLRVT, JVT and JNVT instructions. The operation of the VT flag allows for testing possible overflow conditions at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C: The Carry flag is set to indicate either (i) the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation, or (ii) the state of the last bit shifted out of an operand for a shift. Arithmetic borrow after a subtract operation is the complement of the C flag (i.e, if the operation generated a borrow then C = 0).

PSE: The Peripheral Transaction Server (PTS) Enable bit. Globally enables the PTS when set. Manipulated by the EPTS and DPTS instructions.

I: The global Interrupt bit disables all interrupts except NMI, TRAP and Unimplemented Opcode when cleared.

ST: The STicky bit flag is set to indicate that during a right shift a 1 has been shifted first into the C flag and then shifted out. The ST flag can be used along with the C flag to control rounding after a right shift. The use of the C and ST flags to increase the precision in rounding is described below.

Consider multiplying two 8-bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL
SHR     AX, #4
```

If the C flag is set after the shift, it indicates that the bits shifted off the end of the operand were greater than or equal to one half the least significant bit of the 12-bit result. If the C flag is cleared after the shift, it indicates that the bits shifted off the end of the operand were less than half the LSB of the 12-bit result. Without the ST flag, the rounding decision must be made on the basis of the C flag alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision.

C	ST	Bits Shifted Off
0	0	Value = 0
0	1	0 < Value < 1/2 LSB
1	0	Value = 1/2 LBS
1	1	1/2 LSB < Value < 1 LSB

Imprecise rounding can be major source of error in a numerical calculation; use of the ST flag can increase accuracy.

Interrupt Masks

The lower 8 bits of the PSW individually mask the lower 8 sources of interrupt to the 8XC196MC. Interrupts are discussed in Section 11. These mask bits can be accessed as an SFR (INT_MASK - 08H) in the on-chip lower register file. A separate SFR (INT_MASK1 - 13H) contains the control bits for the higher 8 interrupts. A logical 1 in these bit positions enables servicing of the corresponding interrupt. Bit 9 (I) in the PSW is the global interrupt disable bit. If this bit is cleared, then interrupts are locked out. The I bit is manipulated with the EI and DI instructions. Note that the interrupts are collected in the INT_PEND SFRs (locations 09H and 12H) even if they are locked out. Execution of the corresponding service routines will proceed according to their priorities when they become enabled.

4.5 INSTRUCTION SET

The operations of the 8XC196MC instructions are summarized in Table 4.4. A complete description of each instruction is in Appendix B of this user's guide.

The instructions include a full set of arithmetic and logical instructions for the 8-bit data types (BYTE and SHORT INTEGERS) and for the 16-bit data types (WORD and INTEGERS). The DOUBLE WORD and LONG data types (32 bits) are supported for the product of a 16-by-16 multiply, the dividend of a 32-by-16 divide, and the operand in a shift operation. The remaining operations on 32-bit variables can be implemented by combinations of 16-bit operations. For example, the sequence:

```
ADD    AX,CX
ADDC   BX,DX                ;performs a 32-bit addition
```

and the sequence:

```
SUB    AX,CX
SUBC   BX,DX                ;performs a 32-bit subtraction
```

Operations on REAL (floating-point) variables are not supported directly by the hardware but are supported by the floating-point library (FPAL-96), which implements a single precision subset of the proposed IEEE standard for floating-point operations. The performance of this software is significantly improved by use of the NORML instruction (which normalizes a 32-bit variable) and the ST flag in the PSW (Section 4.4).

In addition to operating on various data types, the 8XC196MC also converts between them. LDBZE (load byte zero extend) converts a BYTE to a WORD, and the LDBSE (load byte sign extend) converts a SHORT INTEGER to an INTEGER. WORDs can be converted to DOUBLE WORDs by simply clearing the upper WORD of the DOUBLE WORD (CLR) and INTEGERs can be converted to LONGs with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction and comparison do not distinguish between signed integers and unsigned words. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. For example, the CMPB (compare byte) instruction is used to compare both signed and unsigned 8-bit quantities. A JH (jump if higher) could be used following the compare for unsigned operands, or a JGT (jump if greater than) could be used for signed operands.

Table 4.5 lists the minimum instruction execution times in terms of state times (state time = one period of CLKOUT = two oscillator periods, Section 14.4). At 16 MHz, one state time equals 125ns. These timings are based on the assumptions listed below the table.

4.6 8XC196MC INSTRUCTION SET ADDITIONS

The 8XC196MC instruction set is based on the set for the 8096BH. This section describes the new instructions that were added.

The new instructions are:

PUSHA	PUSHes the PSW, INT_MASK, INT_MASK1 and WSR.
POPA	POPs the PSW, INT_MASK, INT_MASK1 and WSR.
IDLPD	Sets the part into idle or powerdown mode.
CMPL	Compares 2 long direct values.
DJNZW	Decrement Jump Not Zero using a word counter.
XCH/XCHB	This instruction exchanges two words/bytes, one of which must be within or windowed into the lower register file.
BMOV	Block move using 2 auto-incrementing pointers and a counter.
BMOVI	This instruction operates the same as the block move instruction (BMOV), except BMOVI allows interrupts during the block move.
TIJMP	The table indirect jump, discussed below, allows for indirectly jumping through an address table based on information in an index register and index MASK data.
EPTS	The PTS is enabled following the execution of this instruction.
DPTS	The PTS is disabled following the execution of this instruction.

Table 4.4. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags (Note 2) Z	N	C	V	VT	ST	NOTES
ADD/ADDB	2	D = D + A	↙	↙	↙	↙	↑		
ADD/ADDB	3	D = B + A	↙	↙	↙	↙	↑		
ADDC/ADDCB	2	D = D + A + C	↓	↙	↙	↙	↑		
SUB/SUBB	2	D = D − A	↙	↙	↙	↙	↑		
SUB/SUBB	3	D = B − A	↙	↙	↙	↙	↑		
SUBC/SUBCB	2	D = D − A + C − 1	↓	↙	↙	↙	↑		
CMP/CMPB/CMPL	2	D − A	↙	↙	↙	↙	↑		
MUL/MULU	2	D,D + 2 = D × A							3
MUL/MULU	3	D,D + 2 = B × A							3
MULB/MULUB	2	D,D + 1 = D × A							4
MULB/MULUB	3	D,D + 1 = B ×A							4
DIVU	2	D = (D,D+2)/A,D+2=remainder				↙	↑		3
DIVUB	2	D = (D,D+1)/A,D+1=remainder				↙	↑		4
DIV	2	D = (D,D+2)/A,D+2=remainder				↙	↑		
DIVB	2	D = (D,D+1)/A,D+1=remainder				↙	↑		
AND/ANDB	2	D = D and A	↙	↙	0	0			
AND/ANDB	3	D = B and A	↙	↙	0	0			
OR/ORB	2	D = D or A	↙	↙	0	0			
XOR/XORB	2	D = D (exclusive or) A	↙	↙	0	0			
LD/LDB	2	D = A							
ST/STB	2	A = D							
XCH	2	D ↔ A; D + 1 ↔ A + 1							
XCHB	2	D ↔ A							
BMOV, BMOVI	2	(PTR__HI) + = (PTR__LOW)+; Until COUNT = 0							
LDBSE	2	D = A; D + 1 = Sign (A)							4,5
LDBZE	2	D = A; D + 1 = 0							4,5
PUSH	1	SP = SP − 2; (SP) = A							
POP	1	A = (SP); SP = SP + 2							
PUSHF	0	SP=SP−2;(SP)=PSW;PSW=0;I=0;PSE=0	0	0	0	0	0	0	11
POPF	0	PSW = (SP); SP = SP + 2;I ← ↙	↙	↙	↙	↙	↙	↙	11
PUSHA	0	SP = SP − 2;(SP) = PSW;PSW = 0000h; SP = SP − 2;(SP) = IMASK1/WSR; IMASK1 = 00h; I = 0; PSE = 0	0	0	0	0	0	0	11
POPA	0	IMASK1/WSR = (SP);SP = SP + 2; PSW = (SP); SP = SP + 2	↙	↙	↙	↙	↙	↙	
SJMP	1	PC =PC + 11-Bit-Offset							6
LJMP	1	PC = PC + 16-Bit-Offset							6
BR [Indirect]	1	PC =(A)							
TIJMP	3	PC = ([index] and MASK) 2 + (Table)							
TRAP	0	SP =SP − 2; (SP) = PC; PC = (2010h)							10
SCALL	1	SP = SP − 2; (SP) = PC; PC = PC + 11-Bit-Offset							6

Table 4.4. Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags (Note 2) Z	N	C	V	VT	ST	NOTES
LCALL	1	SP = SP − 2; (SP) = PC; PC = PC + 16-Bit-Offset							6
RET	0	PC = (SP); SP = SP + 2							
J(conditioned)	1	PC = PC + 8-Bit-Offset (If Taken)							6
JC	1	Jump if C = 1							6
JNC	1	Jump if C = 0							6
JE	1	Jump if Z = 1							6
JNE	1	Jump if Z = 0							6
JGE	1	Jump if N = 0							6
JLT	1	Jump if N = 1							6
JGT	1	Jump if N = 0 and Z = 0							6
JLE	1	Jump if N = 1 or Z = 1							6
JH	1	Jump if C = 1 and Z = 0							6
JNH	1	Jump if C = 0 or Z = 1							6
JV	1	Jump if V = 0							6
JNV	1	Jump if V = 1							6
JVT	1	Jump if VT = 1; Clear VT					0		6
JNVT	1	Jump if VT = 0; Clear VT					0		6
JST	1	Jump if ST = 1							6
JNST	1	Jump if ST = 0							6
JBS	3	Jump if Specific Bit = 1							6,7
JBC	3	Jump if Specific Bit = 0							6,7
DJNZ/DJNZW	1	D = D − 1; If D ≠ 0 then PC = PC + 8-Bit-Offset							6
DEC/DECB	1	D = D − 1	↙	↙	↙	↙	↑		
NEG/NEGB	1	D = 0 − D	↙	↙	↙	↙	↑		
INC/INCB	1	D = D + 1	↙	↙	↙	↙	↑		
EXT	1	D = D;D + 2 = Sign(D)	↙	↙	0	0			3
EXTB	1	D = D;D + 1 = Sign (D)	↙	↙	0	0			4
NOT/NOTB	1	D = Logical Not (D)	↙	↙	0	0			
CLR/CLRB	1	D = 0	1	0	0	0			
SHL/SHLB/SHLL	2	C ← msb • • • lsb ← 0	↙	↙	↙	↙	↑		8
SHR/SHRB/SHRL	2	0 → msb • • • lsb → C	↙	↙	↙	0		↙	8
SHRA/SHRAB/SHRAL	2	msb → msb • • • lsb → C	↙	↙	↙	0		↙	8
NORML	2	Left Shift until msb = 1; D = Shift Count	↙	↙	0				8
SETC	0	C = 1			1				
CLRC	0	C = 0			0				
CLRVT	0	VT = 0					0		
RST	0	PC = 2080H	0	0	0	0	0	0	9
DI	0	Disable All Interrupts (I = 0)							
EI	0	Enable All Interrupts (I=1)							
DPTS	0	Disable PTS interrupts (PSE = 0)							
EPTS	0	Enable PTS interrupts (PSE = 1)							
NOP	0	PC = PC + 1							

Table 4.4. Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags (Note 2) Z N C V VT ST	NOTES
SKIP	0	PC = PC + 2		
IPLPD	1	Idle Mode IF Key = 1; Powerdown Mode IF Key = 2 Chip RESET Otherwise		

NOTES:

1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is performed. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Lower Register File; A can be located anywhere in memory.
2. The symbols indicate the effects on the flags:
   - ✓ Cleared or set as appropriate
   - 0 Cleared
   - 1 Set
   - ↑ Set if appropriate; never cleared
   - ↓ Cleared if appropriate; never set
3. D, D+2 are consecutive WORDs in memory; D is DOUBLE-WORD aligned.
4. D, D+1 are consecutive BYTEs in memory; D is WORD aligned.
5. Changes a BYTE to WORD.
6. Offset is a 2's complement number.
7. Specific Bit must be in or windowed into the Lower Register File.
8. The "L" (LONG) suffix indicates DOUBLE-WORD operations.
9. Initiates a RESET by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
10. The assembler does not accept this mnemonic (use the macro file for definition).
11. I = Interrupt Enable (PSW.9).

Table 4.5. Instruction Timing

Instruction	Direct	Immediate	Indirect		Indexed	
			Normal	A-inc	Short	Long
ADD (3op)	5	6	7/10	8/11	7/10	8/11
SUB (3op)	5	6	7/10	8/11	7/10	8/11
ADD (2op)	4	5	6/8	7/9	6/8	7/9
SUB (2op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3op)	5	5	7/10	8/11	7/10	8/11
SUBB (3op)	5	5	7/10	8/11	7/10	8/11
ADDB (2op)	4	4	6/8	7/9	6/8	7/9
SUBB (2op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
CMPL	7					
MUL (3op)	16	17	18/21	19/22	19/22	20/23
MULU (3op)	14	15	16/19	17/20	17/20	18/21
MUL (2op)	16	17	18/21	19/22	19/22	20/23
MULU (2op)	14	15	16/19	17/20	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3op)	12	12	14/17	15/18	15/18	16/19
MULUB (3op)	10	10	12/15	12/16	12/16	14/17
MULB (2op)	12	12	14/17	15/18	15/18	16/19
MULUB (2op)	10	10	12/15	12/16	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3op)	5	6	7/10	8/11	7/10	8/11
AND (2op)	4	5	6/8	7/9	6/8	7/9
OR	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3op)	5	5	7/10	8/11	7/10	8/11
ANDB (2op)	4	4	6/8	7/9	6/8	7/9
ORB	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD	4	5	5/8	6/8	6/9	7/10
ST	4		5/8	6/9	6/9	7/10
XCH	5				8/13	9/14
LDB	4	4	5/8	6/8	6/9	7/10
STB	4		5/8	6/9	6/9	7/10
XCHB	5				8/13	9/14
BMOV	6 + 8 per Word		6 + 11/14 per Word			
BMOVI	7 + 8 per Word +14 For Each Interrupt		7 + 11/14 per Word +14 For Each Interrupt			

Table 4.5. Instruction Timing (Continued)

Instruction	Direct	Immediate	Indirect		Indexed	
			Normal	A-inc	Short	Long
LDBSE, LDBZE	4	4	5/7	6/8	6/8	7/9
PUSH (int)	6	7	9/12	10/13	10/13	11/14
POP (int)	8		10/12	11/13	11/13	12/14
PUSHF (int)	6					
POPF (int)	7					
PUSHA (int)	12					
POPA (int)	12					
PUSH (ext)	8	9	11/14	12/15	12/15	13/16
POP (ext)	11		13/15	14/16	14/16	15/17
PUSHF (ext)	8					
POPF (ext)	10					
PUSHA (ext)	18					
POPA (ext)	18					
LJMP	7					
SJMP	7					
BR [indirect]	7					
TIJMP	15(+ 3 for External Reference)					
TRAP (int)	16					
LCALL (int)	11					
SCALL (int)	11					
RET (int)	11					
TRAP (ext)	18					
LCALL (ext)	13					
SCALL (ext)	13					
RET (ext)	14					
JNST, JST	4/8 Jump Not Taken/Jump Taken					
JNH, JH	4/8 Jump Not Taken/Jump Taken					
JGT, JLE	4/8 Jump Not Taken/Jump Taken					
JNC, JC	4/8 Jump Not Taken/Jump Taken					
JNVT, JVT	4/8 Jump Not Taken/Jump Taken					
JNV, JV	4/8 Jump Not Taken/Jump Taken					
JGE, JLT	4/8 Jump Not Taken/Jump Taken					
JNE, JE	4/8 Jump Not Taken/Jump Taken					
JBS, JBC	5/9 Jump Not Taken/Jump Taken					
DJNZ	5/9 Jump Not Taken/Jump Taken					
DJNZW	6/10 Jump Not Taken/Jump Taken					
CLR, NOT, NEG	3					
DEC, INC	3					
EXT	4					
CLRB, NOTB	3					
DECB, INCB	3					
NEGB	3					
EXTB	4					
NORML	8 + 1 per Shift (9 for 0 Shift)					

Table 4.5. Instruction Timing (Continued)

Instruction	Direct	Immediate	Indirect Normal	Indirect A-inc	Indexed Short	Indexed Long
SHRL	7 + 1 per Shift (8 for 0 Shift)					
SHLL	7 + 1 per Shift (8 for 0 Shift)					
SHRAL	7 + 1 per Shift (8 for 0 Shift)					
SHR	6 + 1 per Shift (7 for 0 Shift)					
SHL	6 + 1 per Shift (7 for 0 Shift)					
SHRA	6 + 1 per Shift (7 for 0 Shift)					
SHRB	6 + 1 per Shift (7 for 0 Shift)					
SHLB	6 + 1 per Shift (7 for 0 Shift)					
SHRAB	6 + 1 per Shift (7 for 0 Shift)					
CLRC	2					
SETC	2					
DI	2					
EI	2					
DPTS	2					
EPTS	2					
CLRVT	2					
NOP	2					
RST	20 (Includes Fetch of CCB/CCB1)					
SKIP	3					
IDLPD	8/25 (Proper Key/Improper Key)					
PTS						
Single Transfer	18	(+ 3 For Ext Reference, +1 If XFER Count = 0)				
Burst Transfer	13	(+ 7 For Each Transfer, 1 Minimum +3 For Each Memory Controller Reference)				
PWM Modes	15					
A/D Soan Mode	21/25					

NOTES:

The timing figures are minimum execution times expressed as state times (one period of CLKOUT = two oscillator periods, Section 14.4) and are based on the following assumptions:

1. The opcode, along with any required operands, have been pre-fetched and reside in the instruction queue.
2. The bus controller operates with the 16-bit bus selected and without wait states for external memory references and pre-fetches. For instructions with indirect or indexed addressing, execution times separated by a slash are for instructions requiring a fetch from internal/external memory.
3. Times for jumps, calls and returns include the 4 state times required to flush the pre-fetch queue and to fetch the opcode at the destination address. This is reflected in the jump taken/not-taken times shown in the table.

4.6.1 Table Indirect Jump (TIJMP)

The primary purpose of the TIJMP is to reduce the interrupt response time associated with servicing multiple interrupt sources which are multiplexed into a single interrupt request line (a single vector).

Normally, to service these interrupts the interrupt service software must sort through status flags to determine the specific source of the interrupt request. This sorting requires a significant number of instruction states and increases the overall interrupt response time.

The TIJMP instruction and the associated address calculation are shown in Table 4.6. The jump table can contain up to 127 entries, and each entry can be any 16-bit jump destination address.

Table 4.6. TIJMP Operation

TIJMP TBASE, [INDEX], INDEX_MASK

TBASE	Word register that points to the bottom of the Jump Table.
INDEX	Word register that points to a 7-bit value to be used as an index into the Jump Table.
INDEX_MASK	An immediate value (7-bit) to be used to mask (AND with) the INDEX value.

ADDRESS CALCULATION

([INDEX] AND INDEX_MASK) * 2 + [TBASE] = Destination

Jump Table

R + x	Dest x
R + 4	
R + 2	
R	

I/O Port 2

5

CHAPTER 5
I/O PORT 2

Port 2 is bidirectional, with CMOS Schmitt Trigger inputs and CMOS level outputs.

Each pin of Port 2 can be set up to perform one of two functions. It can perform standard I/O, or it can carry a special function signal associated with the EPA. If a particular special function signal is not utilized in an application, the associated pin can be used for standard I/O. Note that this selection is made on a pin-by-pin basis, not for the entire port. Thus, each pin of each port can be configured for operation in one of four modes:

1. As a pin for a special function signal controlled by the EPA.
2. As a standard output pin operating in complementary mode.
3. As a standard output pin with open drain, requiring an external pull-up.
4. As a standard high-impedance input pin.

Section 5.1 discusses the basic operation of the port circuits and registers. Sections 5.2 and 5.3 describe how to configure the port pins for the I/O modes and for the special functions.

Standard I/O SFRs. The standard I/O SFRs are listed here for reference. All of these SFRs are byte addressable.

SFR	Address	SFR	Address
		P2_REG	1FD4H
		P2_DIR	1FD2H
P2_PIN	1FD6H	P2_MODE	1FD0H

5.1 CIRCUIT OPERATION AND REGISTERS

The circuit schematic for a single bit of port 2 is shown in Figure 5.1. Signals $\overline{\text{PPU}}$ and $\overline{\text{WKPU}}$ are activated during reset (described in Section 14.5) to initialize the port pins at logical 1. $\overline{\text{PPU}}$ is forced low for about 300 ns by the falling edge of $\overline{\text{RESET}}$ and pulls the pin high. The active low level of $\overline{\text{RESET}}$ then forces $\overline{\text{WKPU}}$ low, which weakly holds the pin high. This transistor is capable of sourcing at least 10 mA. $\overline{\text{WKPU}}$ remains high and the pin remains at logical 1 until the user configures the port pin by writing to the register that selects the port (register Pz_MODE described below).

The port registers are described prior to a discussion of the top portion of the circuit. Specific settings for these registers to realize a desired operation mode are given in Sections 5.2 and 5.3.

P2_MODE, Port 2 mode register. Each bit in P2_MODE determines whether the corresponding pin functions as a standard I/O port pin (bit = 0) or is used for a special function signal (bit = 1).

P2_DIR, Port 2 I/O direction register. Each bit of P2_DIR determines whether the corresponding pin is configured as a complementary output (bit = 0) or as an input or open-drain output (bit = 1).

P2_REG, Port 2 data output register. P2_REG contains data to be driven out by the respective pins. If the CPU writes data to P2_REG while P2_MODE = 0, the effects are seen immediately on the pins. With P2_MODE = 1, the associated peripheral or external component controls the pin. The CPU can still write to P2_REG, but the pin is unaffected until the respective P2_MODE bit is changed to 0. Thus, software can switch P2_MODE to 0, initialize or overwrite the port bit value, and change the P2_MODE bit back to 1. This feature can be used in initialization, fault recovery, exception handling, etc. without having to change the operation of the on-chip peripheral.

P2_PIN, Port 2 pin input register. P2_PIN can be read to determine the current state of the pin, whether the pin is configured for the peripheral signal or for standard output.

SF2_DIR, Port 2 special function direction register. Each port has an additional register SF2_DIR, which is unavailable to the user. When the special function is selected (P2_MODE = 1), SF2_DIR determines whether the special function signal is an output (bit = 0) or an input (bit = 1). When the special function is selected for a pin, the appropriate bit of SF2_DIR is automatically written.

The upper portion of Figure 5.1 shows the logic for driving the output transistor pair Q_U and Q_L. Q_U can source at least −3mA at V_{CC} − 0.7V and Q_L can sink at least 3mA at 0.45V. The guaranteed drive characteristics are specified in the data sheet. Circuit operation in the standard I/O modes and in the special function mode is discussed here. Subsequent sections give instructions for configuring the port pins for the desired modes.

For the standard I/O modes, P2_MODE is 0, which selects the PORT DATA input to the multiplexer. When the user writes to P2_REG, the effect is to drive the PORT DATA input, which drives the P2_REG flip-flop. In the standard I/O mode, the signal SF2_DIR is blocked by the NAND gate. The P2_DIR and P2_REG signals are combined to drive the gates of Q_U and Q_L so that the output P2_PIN is high, low or high impedance (HZ). A logic table for standard I/O operation is given in the left side of Table 5.1. The input mode is the same as the open drain mode with P2_PIN at high impedance (HZ), but no external pull-up is required.

For the special function mode, P2_MODE is 1, which selects the SF SET and SF $\overline{\text{RESET}}$ inputs to the multiplexer. These inputs come from the special function signal (for the case where it is an output signal) and drive the P2_REG flip-flop. The special function signal also automatically determines the value of SF2_DIR (0 for an output signal 1 for an input signal). The logic table for special function operation is in the right side of Table 5.1.

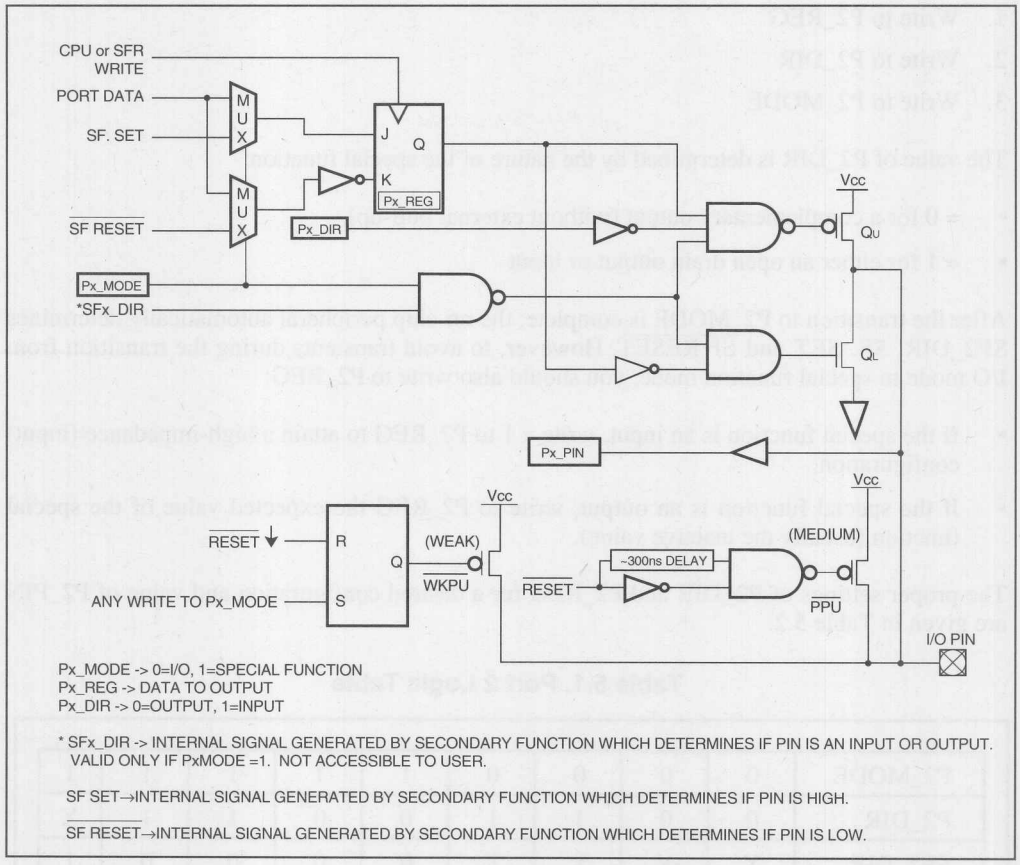


Figure 5.1. Circuit Schematic for Port 2

5.2 CONFIGURATION FOR STANDARD I/O

To configure a port pin for standard I/O, the user writes to the appropriate bits of P2_REG and P2_DIR, and then sets the corresponding bit of P2_MODE to 0. Table 5.2, based on the logic table of Table 5.1, shows the bit values required for a desired configuration. For input mode, the register settings are the same as for open drain output at high impedance, but no external pull-up is required.

5.3 CONFIGURATION FOR SPECIAL FUNCTIONS

To configure a port 2 pin for a special function, the user should write to the port in the following order:

1. Write to P2_REG
2. Write to P2_DIR
3. Write to P2_MODE

The value of P2_DIR is determined by the nature of the special function:

- = 0 for a complementary output (without external pull-up)
- = 1 for either an open drain output or input

After the transition to P2_MODE is complete, the on-chip peripheral automatically determines SF2_DIR, SF. SET and SF RESET. However, to avoid transients during the transition from I/O mode to special function mode, you should also write to P2_REG:

- If the special function is an input, write a 1 to P2_REG to attain a high-impedance (input) configuration.
- If the special function is an output, write to P2_REG the expected value of the special function (usually the inactive value).

The proper settings of P2_DIR and P2_REG for a desired configuration and value of P2_PIN are given in Table 5.2.

Table 5.1. Port 2 Logic Table

P2_MODE	0	0	0	0	1	1	1	1	1
P2_DIR	0	0	1	1	0	0	1	1	X
SF2_DIR	X	X	X	X	0	0	0	0	1
P2_REG	0	1	0	1	0	1	0	1	X
Q_U	off	on	off	off	off	on	off	off	off
Q_L	on	off	on	off	on	off	on	off	off
P2_PIN	0	1	0	HZ*	0	1	0	HZ*	HZ*
Port Config.	Comple-mentary		ODO	ODIO	Comple-mentary		ODO	ODIO	In-put
Port Funct.	Standard I/O				Special Function				

X = Don't Care
HZ = High Impedance
ODO = Open Drain Output
ODIO = Input and Open Drain Output
*During reset and until first write to P2_MODE, WKPU is active.

Table 5.2. Register Settings for Standard I/O Port Operation

I/O Mode	P2_PIN	P2_DIR	P2_REG	QU	QL
Output	0	0	0	off	on
(Complementary)	1	0	1	on	off
Output	0	1	0	off	on
(Open-Drain)	HZ*	1	1	off	off
Input	HZ*	1	1	off	off

Notes: P2_MODE bit = 0
HZ = high impedance
*During reset and until first write to P2_MODE $\overline{\text{WKPU}}$ is active.

Event Processor Array (EPA) 6

CHAPTER 6
EVENT PROCESSOR ARRAY (EPA)

The Event Processor Array (EPA) performs input and output functions associated with two timer/counters. In the input mode the EPA monitors an input pin and looks for an event: a rising edge, a falling edge or an edge in either direction. When the event occurs, the value of the timer/counter is recorded, so that the event is tagged with a time. Recording such an event is called an *input capture*. Input captures are double buffered, which allows two timer captures before an overrun occurs. In output mode the EPA monitors the timer/counter and compares the current contents with a value stored in a register. When a match occurs, the EPA triggers an output event, which can be to: set a pin, clear a pin, toggle a pin or take no action. Such an event is called an *output compare*. The EPA sets an interrupt flag in response to an input capture or an output compare. This flag can optionally cause an interrupt.

The EPA has two 16-bit up/down timer/counters Timer1 and Timer2. Timer1 can be clocked internally or externally. Timer2 can only be clocked internally. The timer/counter is called a *timer* if it is clocked internally and a *counter* if it is clocked externally. The registers that store the time value and control the actions taken comprise a *capture/compare module*. The EPA has four capture/compare modules and four *compare modules* (without capture capability). These modules support event capture, pulse width modulation (PWM) and programmed event generation. The four capture/compare modules are CAPCOMP0-CAPCOMP3 and four compare modules are COMPARE0-COMPARE3. The EPA I/O lines are multiplexed with Port 2..

Figures 6.1 through 6.3 are block diagrams of the timer/counters, a single capture/compare module and the entire EPA structure. The EPA architecture is similar to that of the 80C51FA Programmable Counter Array (PCA) in that it associates capture/compare registers with specific input/output pins. On the other hand, the High-Speed I/O (HSIO) of the 8096 has a CAM and FIFO, which operate with **all** output and input pins, respectively. The 8XC196MC timer/counters, capture/compare and compare modules and the EPA interrupt structure are described further in Sections 6.1 and 6.2.

In the initial documentation Timer1 and Timer2 are called Timer0 and Timer1 respectively. The new names provide uniformity in the advanced 80C196 product family.

EPA and Timer Special Function Registers. All of the EPA registers are accessible as special function registers (SFRs), with the exception of the EPA input buffer registers. The SFRs are listed here with their addresses.

SFR	Address	SFR	Address
TIMER2*	1F7EH	T2CONTROL	1F7CH
TIMER1*	1F7AH	T1CONTROL	1F78H
T1RELOAD	1F72H		
COMP0_CON	1F58H	COMP0_TIME*	1F5AH
COMP1_CON	1F5CH	COMP1_TIME*	1F5EH
COMP2_CON	1F60H	COMP2_TIME*	1F62H
COMP3_CON	1F64H	COMP3_TIME*	1F66H

CAPCOMP0_CON	1F40H	CAPCOMP0_TIME*	1F42H
CAPCOMP1_CON	1F44H	CAPCOMP1_TIME*	1F46H
CAPCOMP2_CON	1F48H	CAPCOMP2_TIME*	1F4AH
CAPCOMP3_CON	1F4CH	CAPCOMP3_TIME*	1F4EH

* These registers must be written as words (16 bits).

External Signals. The external signals associated with the EPA and Timers are:

CAPCOMP0-3 (P2.0-3)	Event signal pins configurable as inputs or outputs
COMPARE0-3 (P2.4-7)	Output event pins
T1CLK (P1.2)	External clock for timer 1
T1DIR (P1.3)	External direction (up/down) for timer 1

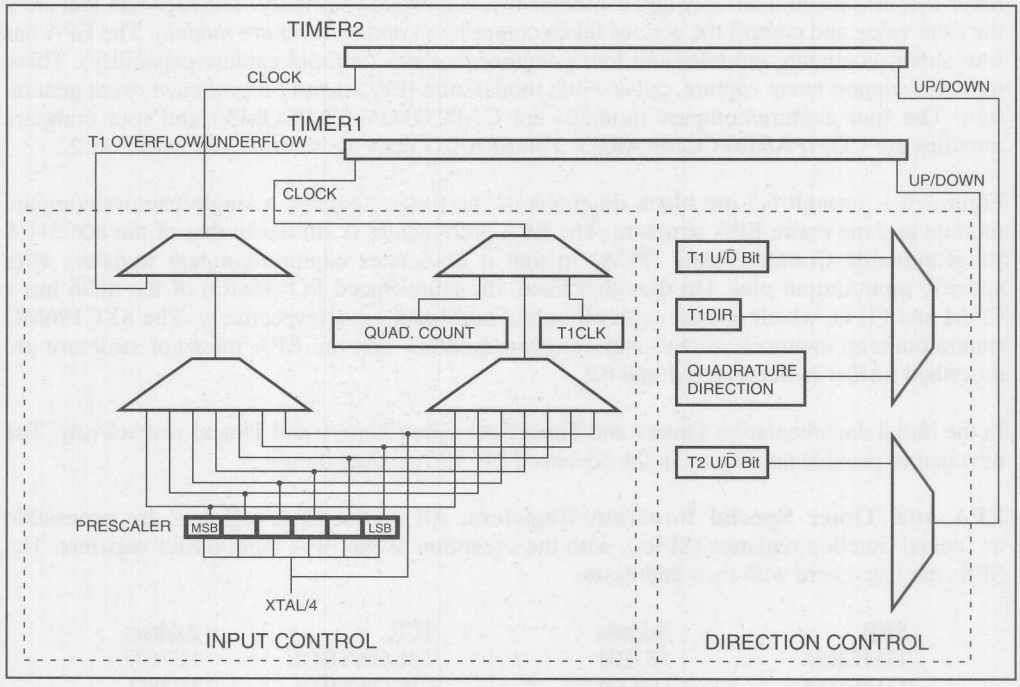


Figure 6.1. EPA Timer/Counters

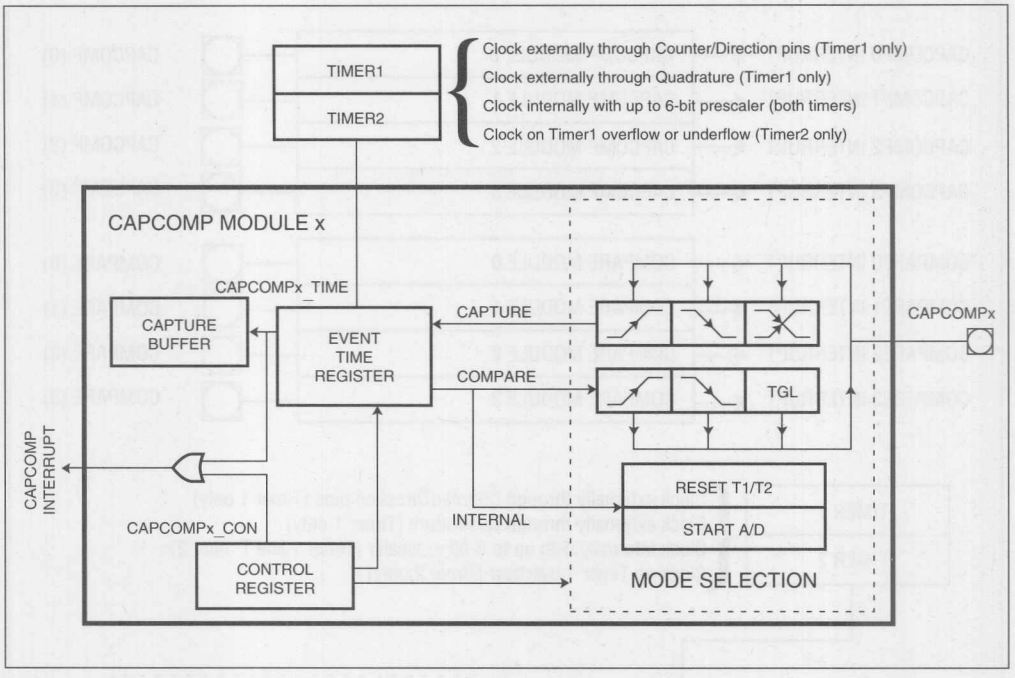


Figure 6.2. A Single EPA Capture/Compare Module

6.1 TIMER/COUNTER STRUCTURE

The EPA timer/counter structure consists of two 16-bit incrementer/decrementers (TIMER1 and TIMER2) and associated control registers (T1CONTROL and T2CONTROL). Each counter overflows from 0FFFFH to 0000H and underflows from 0000H to 0FFFFH. The TxCONTROL registers are defined in Figures 6.4a and 6.4b. The timer/counters can be used as time bases for input captures, output compares and programmed interrupts (software timers). Overflow interrupts from TIMER1 and TIMER2 share one interrupt vector (T1/T2 Overflow/ Underflow Interrupt). The maximum count rate is based on the internal clock rate/2 (XTAL1 frequency ÷ 4). This provides a 250 ns resolution (at 16 MHz) for input capture or output compare. The clock prescaler applies to both internal and external clocks. It consists of a 6-bit counter and 3 bits (P2, P1, P0) which select the source to the timer. This means that not every possible combination of the prescaler is available. The selection is limited to six choices plus one to disable the prescaler (divide by 1). The choices are shown in Figure 6.1 and listed in Figure 6.4.

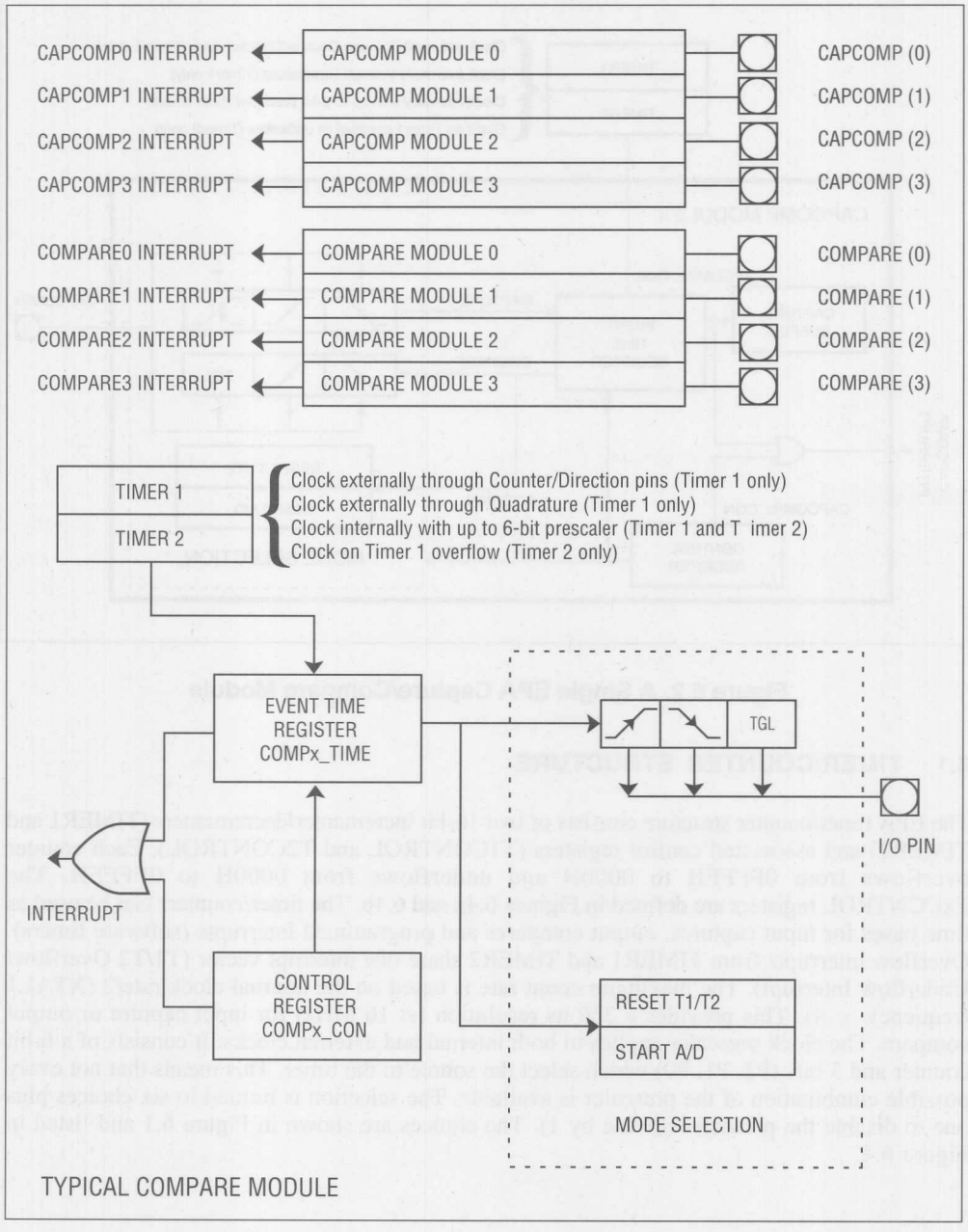


Figure 6.3. EPA Block Diagram

Various combinations of internal and external clock sources and directions are selectable by bits M2, M1, M0. If an external clock is selected, the timer counts on **each** edge (rising and falling) of the clock.

In the quadrature clocking mode, the timer1 input pins (TxCLK and TxDIR) are configured as quadrature inputs. When quadrature encoded signals are input, the timer is incremented or decremented one count per edge. Figure 6.5 illustrates this process, where signals X and Y are derived from T1CLK and T1DIR as shown in Figure 6.6. The count is incremented if the rising/falling edge of T1CLK precedes the rising/falling edge of T1DIR. Conversely, the count is decremented if the rising/falling edge of T1DIR precedes the rising/falling edge of TxCLK. Transitions must be separated by at least 2 state times for proper operation. The count is clocked by PH2, which is PH1 delayed by one-half period. A typical source of quadrature encoded waveforms is a shaft angle encoder, as illustrated in Figure 6.6.

6.2 CAPTURE/COMPARE STRUCTURE

The EPA structure (Figure 6.3) contains four capture/compare modules, each of which supports high speed I/O for a single pin. In addition, there are four compare modules which support high speed output for a single pin. Each EPA module can generate an interrupt, reset its own time base timer, start an A/D conversion, generate a reload trigger for the wareform generator, change the output pin state or reset its opposite time base timer. The capture/compare modules can also capture time base timer values based on transitions on an input pin.

6.2.1 Capture/Compare Modules

Each Capture/Compare Module (Figure 6.2) handles the input and output functions for a single pin. It contains an event-time register (CAPCOMPx_TIME), which is double buffered for input captures, a control register (CAPCOMPx_CON) and the associated control logic. CAPCOMPx_TIME is a 16-bit register that contains the scheduled time for a programmed event (output to a pin or internal control) or captured time from an EPA input event. Figure 6.7 shows the functions specified by CAPCOMPx_CON.

Figure 6.8 gives examples of operations selected by several CAPCOMPx_CON settings.

The PFE bit allows you to use the EPA activities to start an A/D conversion and/or cause a reload in the waveform generator (see Chapter 7 for WG reload).

When PFE is set and an EPA activity occurs, depending on the EPA module, it can start conversion on a selected A/D channel, or cause the reload of new values in the waveform generator.

CAPCOMP0, CAPCOMP2, COMPARE0 or COMPARE2 control the WG reload, but CAPCOMP1, CAPCOMP3, COMPARE1 or COMPARE3 control the A/D conversion. These activities take place in addition to the primary activity of the EPA module, i.e., toggling the output pin, generating interrupt, capturing, etc.

Output Compare Operation. A single event-time register and control register pair is used together with a timer for output compare generation. The event is programmed by first writing CAPCOMPx_CON and then loading CAPCOMPx_TIME. When the contents of CAPCOMPx_TIME matches the time value of the specified timer, the action in CAPCOMPx_CON is executed. If the Re-Enable (RE) bit is not set, the event is automatically disabled. At this point, to execute another output event, CAPCOMPx_TIME and optionally CAPCOMPx_CON must be written. If the RE bit is set, the event is not disabled and will re-execute on the next match with the timer. To generate an output compare, CAPCOMPx_TIME must always be written following a write to CAPCOMPx_CON.

T1CONTROL (IF78H)

7	6	5	4	3	2	1	0
CE	UD	M2	M1	M0	P2	P1	P0

CE: = 0 disable timer, = 1 enable timer (From Reset the timers are disabled and not free running.)
UD: = 0 count down, = 1 count up

M2	M1	M0	Clock Source	Clock Direction
0	0	0	internal	UD bit
0	0	1	external	UD bit
0	1	0	internal	external pin
0	1	1	external	external pin
1	0	0	reserved	
1	1	0	reserved	
1	1	1	quadrature*	

*quadrature clocking mode 1 (P0 = P1 = P2 = 0)
quadrature clocking mode 2 (P0 = P1 = P2 = 1)
The counter input clock and the count direction is taken from the quadrature clocking circuit. (See Figure 6.5). In mode 2 Timer1 is preset on underflow using the T1_RELOAD register.

P2	P1	P0	Input Divide By	Resolution (at 16 MHz)
0	0	0	1	250 ns
0	0	1	2	500 ns
0	1	0	4	1 µs
0	1	1	8	2 µs
1	0	0	16	4 µs
1	0	1	32	8 µs
1	1	0	64	16 µs
1	1	1	1	enable T1_RELOAD

Figure 6.4a. T1CONTROL Register

An interrupt pending bit is set each time a match occurs on an enabled event (even if no output compare or internal event is specifically selected). This bit can be masked out in the PI_MASK register. If the interrupt is masked, software can still determine that an event has executed by polling the corresponding interrupt pending flags.

T2CONTROL (IF7CH)

7	6	5	4	3	2	1	0
CE	UD	M2	M1	M0	P2	P1	P0

CE: = 0 disable timer, = 1 enable timer (From Reset the timers are disabled and not free running.)
UD: = 0 count down, = 1 count up

M2	M1	M0	Clock Source	Clock Direction
0	0	0	internal	UD bit
0	0	1	reserved	
0	1	0	reserved	
0	1	1	reserved	
1	0	0	T1 over/underflow	UD bit
1	1	0	T1 over/underflow	Timer1
1	1	1	reserved	

P2	P1	P0	Input Divide By	Resolution (at 16 MHz)
0	0	0	1	250 ns
0	0	1	2	500 ns
0	1	0	4	1 µs
0	1	1	8	2 µs
1	0	0	16	4 µs
1	0	1	32	8 µs
1	1	0	64	16 µs
1	1	1	reserved	

Figure 6.4b. T2CONTROL Register

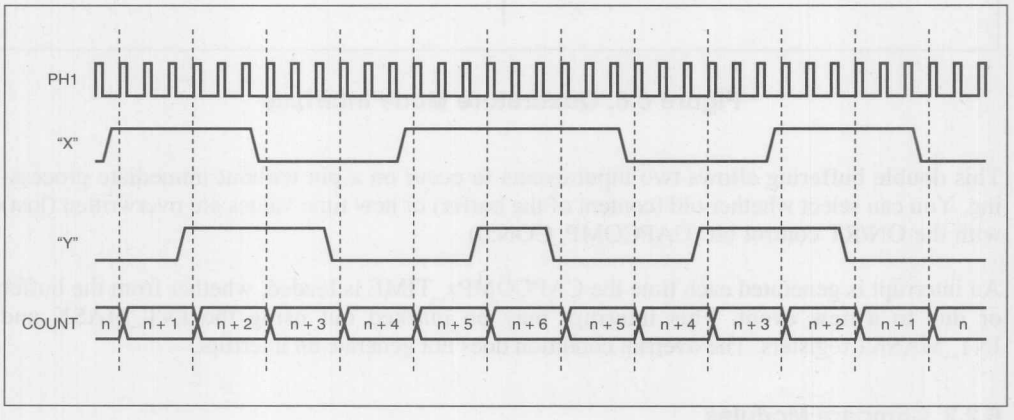


Figure 6.5. Quadrature Timing Mode

Input Capture Operation. In addition to the event-time register and control register pair used for the output compare function, an event-time buffer register provides buffering of input events until they can be handled by software or PTS action. This buffer is the only EPA register that is not an SFR. An input capture involves two registers: CAPCOMPx_TIME and its buffer. When the event occurs, the corresponding timer value is loaded into the buffer. The buffer is then loaded into CAPCOMPx_TIME when CAPCOMPx_TIME is empty. This

provides double buffering of all input captures. An interrupt pending bit is set each time CAPCOMPx_TIME is loaded. If the buffer contains data, and the PTS is used to read CAPCOMPx_TIME, then the two PTS transfers will occur almost back-to-back, i.e., with one instruction executed between the transfers. Unwanted values in CAPCOMPx_TIME must be read in order to allow capture of subsequent events and to set the interrupt pending bit.

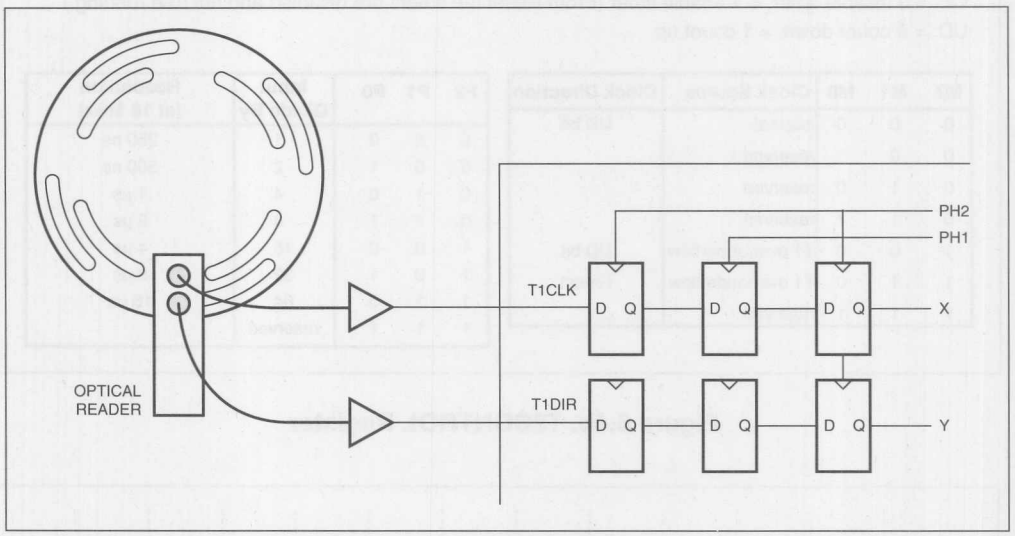


Figure 6.6. Quadrature Mode Interface

This double buffering allows two input events to occur on a pin without immediate processing. You can select whether old (content of the buffer) or new time values are overwritten (lost) with the ON/RT control bit (CAPCOMP_CON.0)

An interrupt is generated each time the CAPCOMPx_TIME is loaded, whether from the buffer or due to a new event. This interrupt may be masked out using the INT_MASK and INT_MASK1 registers. The overrun condition does not generate an interrupt.

6.2.2 Compare Modules

The four EPA compare modules (Figure 6.3) are nearly identical to the capture/compare modules with the capture function removed. Each compare module handles all of the same compare-related functions: A/D starts, WG reload, timer reset and timer interrupts. It contains an event-time register (COMPx_TIME), a control register (COMPx_CON) and control logic. COMPx_TIME is functionally identical to CAPCOMPx_TIME. It contains the time at which an event is to occur (16-bit value). COMPx_CON is similar to CAPCOMPx_CON and is shown in Figure 6.9. Example COMPx_CON settings are shown in Figure 6.10.

Compare modules cannot perform input capture. The compare modules share output pins with P2.4-P2.7.

CAPCOMPx_CON*

7	6	5	4	3	2	1	0
TB	CE	M1	M0	RE	PFE	ROT	ON/RT

TB Time base select. TB = 0 for TIMER1, = 1 for TIMER2.
CE Compare enable. CE = 1 for Compare mode, = 0 for capture
M1, M0:

M1	M0	Capture Mode	Compare Mode
0	0	No capture (no operation)	No compare
0	1	Capture - edge	Output 0
1	0	Capture + edge	Output 1
1	1	Capture ± edge	Toggle output

RE Reenable. For RE=1 the compare function is always enabled; for RE=0 the compare function will drive the output only once, and then CAPCOMPn_TIME must be rewritten.
PFE Peripheral Function Enable. Enable A/D Conversion or WG Reload.
1 = PFE enable 0 = no action
CAPCOMP0_CON and CAPCOMP2_CON control WG Reload
CAPCOMP1_CON and CAPCOMP3_CON control A/D conversion
ROT Reset Opposite Timer:

ROT	Capture Mode	Compare Mode
0	Disable ROT function	Selects time base (TB) for possible reset
1	Resets opposite timer (The one not used for capture)	Selects opposite time base (Not TB) for possible reset

ON/RT Overwrite New/Reset Timer:

ONRT	Capture Mode	Compare Mode
0	New Data lost on overrun	Disables ON/RT function
1	Old data of CCX_BUF lost on data overrun	Reset time base determined by TB and ROT

Figure 6.7. CAPCOMPx_CON Register

TB	CE	M1	M0	RE	PFE	ROT	ON/RT	Operation
CAPTURE								
X	0	0	0	–	–	–	–	No Operation (no interrupt)
X	0	1	0	–	X	X	X	Capture on + Edges
X	0	0	1	–	X	X	X	Capture on − Edges
X	0	1	1	–	X	X	X	Capture on ± Edges
X	0	X	1	–	X	1	X	Reset Opposite Timer
X	0	1	X	–	X	1	X	Reset Opposite Timer
X	0	X	1	–	1	X	X	Peripheral Function Enable
X	0	1	X	–	1	X	X	Peripheral Function Enable
X	0	X	1	–	X	X	X	Generate Interrupt
X	0	1	X	–	X	X	X	Generate Interrupt
COMPARE								
X	1	0	1	X	X	X	X	Reset Output Pin "0"
X	1	1	0	X	X	X	X	Set Output Pin "1"
X	1	1	1	X	X	X	X	Toggle Output Pin
X	1	X	X	X	X	0	1	Reset Same Timer
X	1	X	X	X	X	1	1	Reset Opposite Timer
X	1	X	X	X	1	X	X	Peripheral Function Enable
X	1	X	X	X	X	X	X	Generate Interrupt
X	1	0	0	X	0	X	0	Generate Interrupt Only (Software Timer)

–: bit is not used
X: bit may be used, but does not affect function described

Figure 6.8. Example CAPCOMPx_CON Operations

COMPx_CON

7	6	5	4	3	2	1	0
TB	CE	M1	M0	RE	PFE	ROT	ON/RT

TB Time base select. TB = 0 for TIMER1, = 1 for TIMER2
CE Compare enable CE = 1 enable compare mode CE = 0 disable compare mode (interrupt will not occur)

M1, M0:

M1	M0	Compare Mode
0	0	No output
0	1	Output 0
1	0	Output 1
1	1	Toggle output

RE Reenable. For RE=1 the compare function is always enabled; for RE=0 the compare function will drive the output only once, and then COMPx_TIME must be rewritten.
PFE Peripheral Function Enable. Enable A/D Conversion or WG Reload.
1 = PFE enable 0 = no action
COMP0_CON and COMP2_CON controls WG Reload
COMP1_CON and COMP3_CON CONTROLS A/D conversion
ROT Reset Opposite Timer:

ROT	Compare Mode
0	Selects time base (TB) for possible reset
1	Selects opposite time base (The one not used for capture) for possible reset

ON/RT Overwrite New/Reset Timer:

ON/RT	Compare Mode
0	Disables ON/RT function
1	Reset time base determined by ROT

Figure 6.9. EPA Compare Control (COMPx_CON)

TB	CE	M1	M0	RE	AD	ROT	ON/RT	Operation
COMPARE								
X	0	X	X	–	–	–	–	No Operation (no interrupt)
X	1	0	1	X	X	X	X	Reset Output Pin "0"
X	1	1	0	X	X	X	X	Set Output Pin "1"
X	1	1	1	X	X	X	X	Toggle Output Pin
X	1	X	X	X	X	0	1	Reset Same Timer
X	1	X	X	X	X	1	1	Reset Opposite Timer
X	1	X	X	X	1	X	X	Peripheral Function Enable
X	1	X	X	X	X	X	X	Generate Interrupt
X	1	0	0	X	0	X	0	Generate Interrupt Only (Software Timer)

–: bit is not used
X: bit may be used, but does not affect function described

Figure 6.10. Example COMPX_CON Operations

Waveform Generator 7

CHAPTER 7
WAVEFORM GENERATOR

The Waveform Generator (WG) produces 3 pairs of complimentary PWM signals with minimum CPU involvement. Each signal is independently programmable. A dead-time generator and phase inverter circuit provides non-overlapping signals for each PWM output pair. This peripheral is optimized for controlling 3-phase induction AC motors. It can also control brushless DC motors and other devices requiring multiple PWM outputs. An on-chip protection circuit shuts off all 6 outputs in response to an external signal.

All 3 PWM waveforms have the same period and dead-time. With a 16 MHz clock, the period is 0.25 μs to 16 μs for centered PWM and 0.125 μs to 8 ms for edge triggered PWM. The dead-time is adjustable from 0.125 μs to 125 μs. Each pair of PWM signals can be adjusted in 0.25 μs increments for centered PWM, or 0.125 μs increments for edge triggered PWM.

The time base for the waveform generator is a 16-bit up/down counter with a 16-bit reload register. Each phase has its own 16-bit compare register and 10-bit dead-time generator. Figure 7.1 shows a block diagram of the waveform generator.

Waveform Generator Pins. Six output pins are associated with WG. These pins can be programmed to output 3 phase complementary PWM, V_{CC} or V_{SS}. When set to V_{CC} or V_{SS}, they act as standard output ports. The EXTINT pin is used as input to the protection circuit of WG. If not used for the protection circuit, it may be used as the external interrupt pin.

$\overline{WG1}$ and WG1	(P6.0-1)	Phase 1 signals
$\overline{WG2}$ and WG2	(P6.2-3)	Phase 2 signals
$\overline{WG3}$ and WG3	(P6.4-5)	Phase 3 signals
EXTINT		

7.1 WG Special Function Registers

All of the waveform generator registers are read/write registers, except for WG_COUNT, which is a read only register. The SFRs are listed here with their addresses.

SFR	Address	Function
WG_OUT	1FC0H	Controls output format
WG_COMP1	1FC2H	Phase 1 compare register
WG_COMP2	1FC4H	Phase 2 compare register
WG_COMP3	1FC6H	Phase 3 compare register
WG_RELOAD	1FC8H	16-bit reload register
WG_COUNT	1FCAH	16-bit counter (*read only)
WG_CON	1FCCH	WG mode control and 10-bit dead time
WG_PROTECT	1FCEH	Protection circuit control register

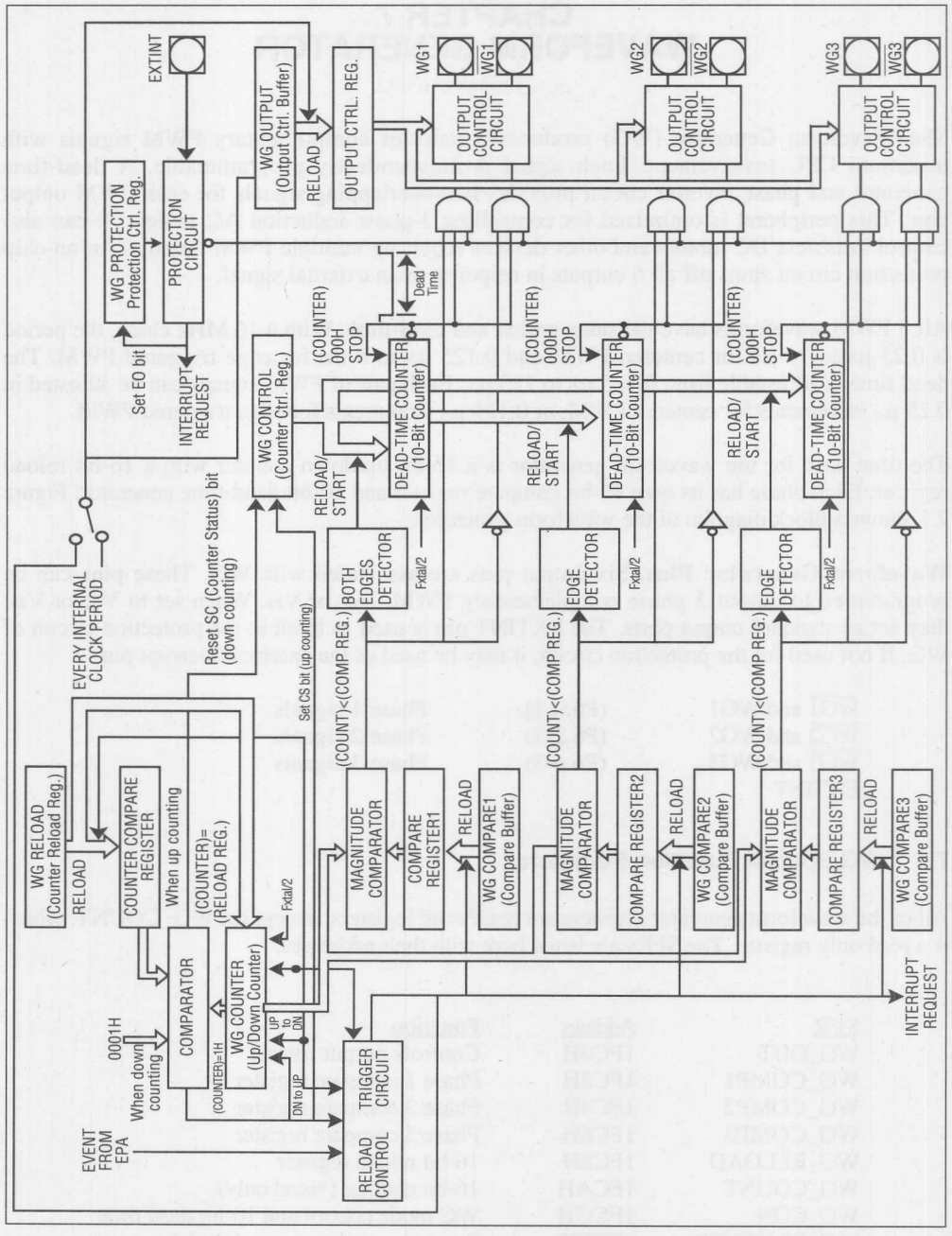


Figure 7.1. Block Diagram of WG

7.1.1 Up/Down Counter (WG_COUNT)

The 16-bit up/down counter is the time base for the three complementary signals. Its clock source is the oscillator frequency divided by two. Thus, it increments or decrements once each state time. The counter can be read at all times, but cannot be directly written. The user writes to a buffer register called WG_RELOAD and the value periodically gets loaded to the counter.

7.1.2 Reload Register (WG_RELOAD)

WG_RELOAD is actually a pair of 16 bit registers. When you read or write WG_RELOAD you access WG_RELOAD. The value written to WG_RELOAD, periodically (depending on the mode of operation), is loaded into the second register. The second register, which is called the Counter Compare register, is the time register that WG_COUNT is actually compared to. The loading of the second register takes place when WG_COUNT = 1 or when it is equal to the value in the Counter Compare register, depending on the WG mode. If you write 0 to WG_RELOAD, when the value is loaded into the counter, it stops counting.

7.1.3 Phase Compare Buffer Registers (WG_COMPx)

There are three 16-bit Phase Compare buffers (x = 1, 2 or 3) that can be read and written. Each phase compare buffer has an associated compare register whose value is compared to WG_COUNT on every count. These phase compare registers cannot be directly accessed by the user.

When WG_COUNT is stopped, a value written to the phase compare buffer is loaded into its phase compare register half a state time later. When WG_COUNT is running, data transfer depends on the WG mode. See Section 7.2.

The reload signal for the compare registers is also used as an interrupt request signal to the CPU in both up and up/down counting modes.

7.1.4 WG Control Register (WG_CON)

Ten bits of this register form a reload register for all three 10-bit dead-time generators. Each dead-time generator has a 10-bit counter. The dead-time generated is equal to the 10-bit value in the dead-time reload register in state times. The dead-time can be changed at anytime using the dead-time reload register. WG_CON also contains the mode control bits. Figure 7.2 shows all the bit names and their function.

7.1.5 Output Control Buffer Register (WG_OUT)

This 16-bit register selects what goes out on the output pins. It allows the user to define and program the active state of each pin independently. The register can be accessed as a word or as 2 separate bytes. The actual output control register is not accessible to the user. WG_OUT is a buffer for user inputs to the output control register. See Figure 7.3.

WG_CON (1FCCH)

15	14	13	12	11	10	9 - 0
0	0	M1	M0	CS	EC	D9-D0

0: Reserved, write 0.

M1, M0: Mode bits, see table 7.1.

CS: Counter Status, 1 = up counting, 0 = down counting

EC: Disable/Enable Counter (WG_COUNT), 1 = enable counting, 0 = disable counting

D9-D0: 10-bit dead-time reload register

Figure 7.2. WG Control Register

When the SYNC bit is 0, updates to WG_OUT are loaded into the output control register immediately. If the SYNC bit is 1, updates to WG_OUT are synchronized with the events listed in Table 7.1.

You must initialize WG_OUT with the SYNC bit = 0 before attempting to synchronize the outputs to the other events. After the SYNC bit is set, writes to WG_OUT are buffered except for the SYNC bit. The SYNC bit is not buffered, it changes immediately.

WG_OUT also holds the bits that control the two 8-bit PWM pins (pins P6.6 and P6.7). This facilitates controlling all 8 outputs of Port 6 with a single word. The register is configured so the port can easily be used as an output port by only writing to the lower byte.

7.2 WG MODES OF OPERATION

There are two bits in WG_CON that select the WG operation. Bits M0 and M1 control when the new values are loaded. These bits also program WG_COUNT to count up, or count up and down. Table 7.1 shows these modes of operation.

When the up count mode is selected, updates to WG_RELOAD, WG_COMPx and WG_OUT are synchronized with the beginning of the period or an EPA event. The output, in this mode, is called edge aligned PWM. This is the normal PWM output.

When the up/down mode is selected, the reload of the new values are synchronized with the center and optionally the beginning of the PWM period. This output is called center aligned PWM. A centered PWM waveform is symmetrical around the middle of the PWM output. In this mode you can update the control registers once or twice per PWM period. The output signal carries much less harmonic contents than the edge aligned PWM

WG_OUT (1FC0H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP1	OP0	SYNC	PE7	PE6	PH32	PH22	PH12	P7	P6	PH31	PH30	PH21	PH20	PH11	PH10

OP1: Output Polarity (for positive-phase outputs)
1 = P6.1, P6.3 and P6.5 are active-high
0 = P6.1, P6.3 and P6.5 are active-low

OP0: Output Polarity (for negative-phase outputs)
1 = P6.0, P6.2 and P6.4 are active-high
0 = P6.0, P6.2 and P6.4 are active-low

SYNC: Synchronizes loading of Output Control register, 1 = synchronizes the load according to Table 7.1, 0 = loads immediately

PE7, PE6: Enable P6.7, P6.6 (respectively) as PWM output

P7, P6: P6.7, P6.6 (respectively) output value

n = 1, 2 or 3			WGn		$\overline{WGn}$	
PHn2	PHn1	PHn0	OP1 = 1	OP = 0	OP0 = 1	OP0 = 0
0	0	0	Low	High	Low	High
0	0	1	Low	High	High	Low
0	1	0	High	Low	Low	High
0	1	1	High	Low	High	Low
1	0	0	Low	High	Low	High
1	0	1	Low	High	$\overline{WG}$	WG
1	1	0	$\overline{WG}$	WG	Low	High
1	1	1	$\overline{WG}$	WG	WG	$\overline{WG}$

PEn	Pn	Pin Output
0	0	0
0	1	1
1	X	PWM Output

PH12 PH11 PH10 --. P6.0 and P6.1 control bits
PH22 PH21 PH20 --. P6.2 and P6.3 control bits
PH32 PH31 PH30 --. P6.4 and P6.5 control bits

Figure 7.3. WG Output Control Register (WG_OUT)

For the rest of this chapter we will refer to the modes by their numbers, i.e., 0, 1, 2 or 3. Table 1 shows the modes of operation and the associated reload condition for each mode.

In this context the term "reload" means transfer of data from WG_RELOAD to WG_COUNT, from WG_COMPx to the compare registers counter, and from WG_OUT to the Output Control buffer.

To enable the WG interrupt which is in the PI_MASK/PI_PEND register, the peripheral interrupt bit (PI) in the INT_MASK/INT_PEND register must be set. See Section 11.1.1.

7.3 WAVEFORM GENERATING PROCESS

When enabled, WG_COUNT continuously counts up, or up and down depending on what mode is selected. The contents of WG_COUNT, at every count, is compared to the value of the Counter Compare register as well as all three phase compare registers. A match between WG_COUNT and any of these registers generates various activities. The type of activity depends on which register matches with WG_COUNT, and what WG mode is selected.

Table 7.1. WG Modes of Operation

Mode	Mode bits	PWM Type	Counter Operation	Reload Condition For WG_COMPx	Reload Condition For WG_OUT*
0	0 0	Center	Up/Down	WG_COUNT match	WG_COUNT match
1	0 1	Center	Up/Down	WG_COUNT match or WG_COUNT = 1	WG_COUNT match or WG_COUNT = 1
2	1 0	Edge	Up	WG_COUNT match	WG_COUNT match
3	1 1	Edge	Up	WG_COUNT match or EPA Event	EPA Event

*These reload trigger signals are enabled when the SYNC bit in the output control register (WG_OUT) is 1. Otherwise, WG_OUT is loaded whenever it is written.

First we will look at a match between WG_COUNT and the counter compare register.

In mode 0 the counter counts up and down. A match causes the following events to take place:

1. Load the counter (WG_COUNT) with the contents of WG_RELOAD.
2. Load the counter compare register with the contents of WG_RELOAD.
3. Load phase compare register x (x = 1, 2 or 3) with the contents of WG_COMPx.
4. Load WG_OUT with the contents of the output buffer register.
5. Set the WG interrupt flag in the PI_PEND register.

After a new (or old) value is loaded into WG_COUNT, it counts down until it reaches 0001, waits for one additional state time, then counts up to the value of the Counter Compare register.

In mode 1 a match causes the same sequence as in mode 0. In addition, when WG_COUNT = 0001 steps 2 through 4 are repeated.

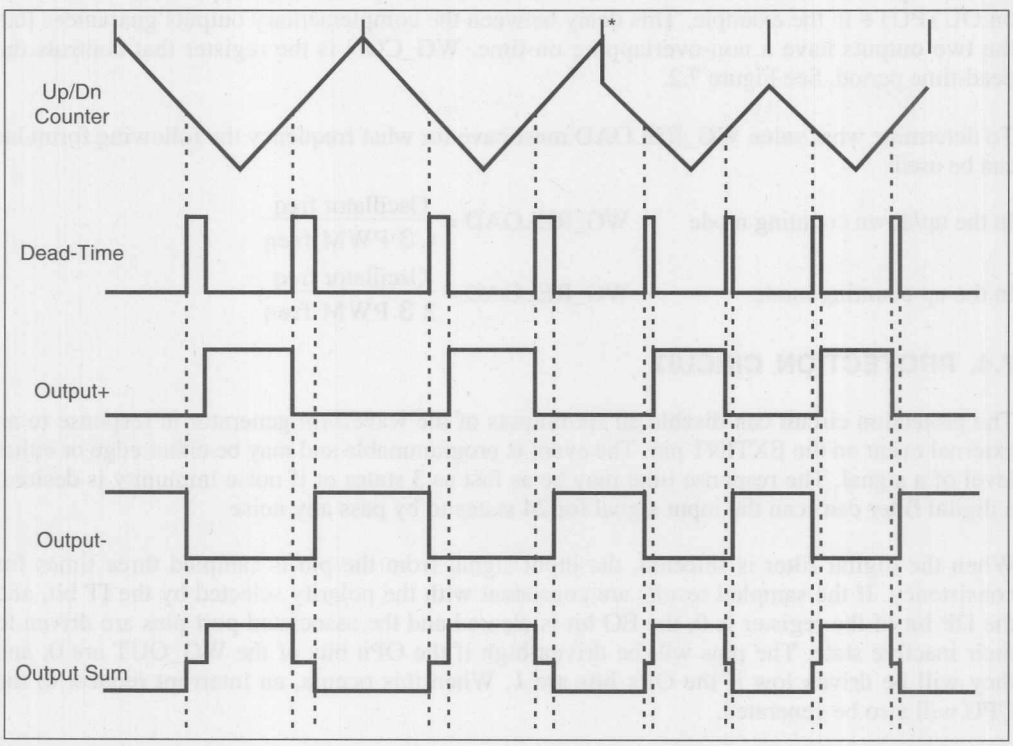
The second reload allows the PWM duty cycle and frequency to be changed in the center of the period.

In mode 2 the counter counts up only. A match loads up WG_COUNT with 0001 and up counting continues.

1. Load the Counter Compare register with the contents of WG_RELOAD.
2. Load Phase Compare register x (x = 1, 2 or 3) with the contents of WG_COMPx.
3. Load WG_OUT with the contents of the output buffer registers.
4. Set the WG interrupt flag in the PI_PEND register.

Mode 3 differs from mode two in the following ways: 1) A match does not cause the reload of WG_OUT. 2) An EPA event causes the reload of WG_OUT and all the activities caused by a match in mode 2.

You may write to WG_RELOAD at any time, but the value transfers to WG_COUNT (and counter compare register) only under the conditions mentioned above.



*Shows a change of value in the reload register

Figure 7.4. Waveforms of Events

Now we will look at what happens when there is a match between WG_COUNT and any of the phase compare registers. The circuitry for all three phases are identical, therefore, we will

analyze the effects of the match on one of the phases. The circuitry can be divided into 3 sections:

1 Counter compare section consists of WG_COUNT, WG_RELOAD, phase compare registers and comparators.

2. Dead-time generator and phase inverter section contains 10-bit counter, phase inverter circuitry.

3. Output control section contains output control registers.

The activity starts in the counter compare section where the match occurs. The match generates a signal that triggers the dead-time circuitry. This signal starts the dead-time counter and simultaneously produces one of the output signals. In the example of Figure 7.3 it turns off OUTPUT]. When the dead-time counter times out the complementary output switches. It turns on OUTPUT+ in the example. This delay between the complementary outputs guarantees that the two outputs have a non-overlapping on-time. WG_CON is the register that controls the dead-time period. See Figure 7.2.

To determine what value WG_RELOAD must have for what frequency the following formulas can be used:

In the up/down counting mode $$WG\_RELOAD = \frac{\text{Oscillator freq}}{4\ 3\ \text{PWM freq}}$$

In the up counting mode $$WG\_RELOAD = \frac{\text{Oscillator freq}}{2\ 3\ \text{PWM freq}}$$

7.4. PROTECTION CIRCUIT

The protection circuit can disable all six outputs of the waveform generator in response to an external event on the EXTINT pin. The event is programmable and may be either edge or either level of a signal. The response time may be as fast as 3 states or if noise immunity is desired, a digital filter can scan the input signal for 24 states to by pass any noise.

When the digital filter is selected, the input signal from the pin is sampled three times for consistency. If the sampled results are consistent with the polarity selected by the IT bit, and the DP bit of the register is 0, the EO bit is cleared and the associated port pins are driven to their inactive state. The pins will be driven high if the OPn bits of the WG_OUT are 0, and they will be driven low if the OPx bits are 1. When this occurs, an interrupt request to the CPU will also be generated.

P6.0 through P6.5 are weakly held high during and after reset until the EO bit of the WG_OUT register is set.

The protection circuit consists of the protection circuit control register (WG_PROTECT), a sampling circuit and a sampling clock generator. Figure 7.5 shows the bits of WG_PROTECT, and Figure 7.6 shows the protection circuit diagram.

WG_PROTECT (1FCEH)

7	6	5	4	3	2	1	0
0	0	0	0	ES	IT	DP	EO

0: Reserved, write 0

ES: Enable Sampling circuitry
- 1 = protection/interrupt triggered by sampling
- 0 = protection/interrupt triggered by edge

IT: Interrupt Type control bit
- 1 = rising edge/high trigger
- 0 = falling edge/low trigger

DP: Disable/Enable Protection circuit
- 1 = disable protection circuit
- 0 = enable protection circuit

EO: Enable/Disable Output. Must be set when port is used as output
- 1 = enable output
- 0 = disable output

Figure 7.5. WG Protection Control Register

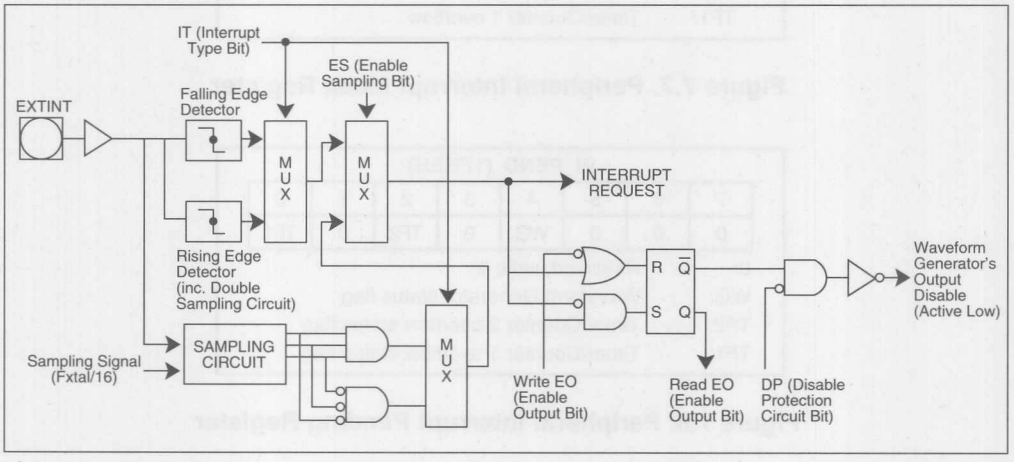


Figure 7.6. WG Protection Block Diagram

7.5. WG INTERRUPT

Two interrupts are associated with the waveform generator

WG Interrupt

Interrupt generated from the protection circuit input (EXTINT)

When a valid input is applied to EXTINT pin, it triggers the protection circuit, which disables all output pins, and generates an interrupt. The interrupt vector is the same as EXTINT vector.

Every time a reload takes place in WG_Count, an interrupt can be generated. The interrupt mask bit must be set to enable WG interrupt. This bit is in the peripheral interrupt mask register (PI_MASK). Figure 7.7 shows this register.

The peripheral interrupt pending register (PI_PEND) shown in Figure 7.8 is a read only register. The reserved bits will always read 1.

CAUTION

Reading PI_PEND clears all bits. Therefore, the value of the register must be stored in a shadow register if other interrupts other than WG are used.

PI_MASK (1FBCH)

7	6	5	4	3	2	1	0
0	0	0	WG	0	TF2	0	TF1

0: Reserved, write 0
WG: Waveform Generator
TF2: Timer/Counter 2 overflow
TF1: Timer/Counter 1 overflow

Figure 7.7. Peripheral Interrupt Mask Register

PI_PEND (1FBEH)

7	6	5	4	3	2	1	0
0	0	0	WG	0	TF2	0	TF1

0: Reserved, write 0
WG: Waveform Generator status flag
TF2: Timer/Counter 2 overflow status flag
TF1: Timer/Counter 1 overflow status flag

Figure 7.8. Peripheral Interrupt Pending Register

7.6 Applications Examples

The waveform generator can support several kinds of motors. The following examples are shown in this section:

180° type inverter (Induction motor inverter with centered PWM)

120° type inverter (DC brushless motor)

Stepper motor driven by switched PWM

The first example is for a 180° inverter controlling a three phase induction motor.

The value contained in the counter reload register (WG_RELOAD) is set to half of the PWM period, the up/down counting mode will be used to generate the full period. The counter control register (WG_CON) is programmed for either mode 0 or mode 1 operation. When mode 0 is selected, the value of each phase compare buffer (WG_COMPx) is reloaded into the associated register at the beginning of every PWM period. This corresponds to modulation being performed using a saw-tooth waveform. When mode 1 is selected each phase compare buffer will be reloaded into the associated register every half of a PWM period. This method is equivalent to using a triangle wave as the carrier and is called "triangle wave modulation" or "double edge modulation."

When the phase compare register reload signal occurs, an interrupt request is generated. The new phase compare buffer and the counter reload register values can then be calculated and loaded in the interrupt service routine. The 10-bit dead time value controls the dead time between complementary PWM outputs.

To generate centered PWM, set the output mode bits (PHx2, PHx1, PHx0) in the output mode control register (WG_OUTPUT) to 1. The positive-phase waveforms (WG1, WG2, WG3) and the negative-phase waveforms ($\overline{WG1}$, $\overline{WG2}$, $\overline{WG3}$ are output on P6.0-P6.5. Figure 7.9 shows WG1 and $\overline{WG1}$ in mode 1. Other phase waveforms are equivalent to that of these waveforms.

The second example is a 120° inverter for controlling a three phase DC brushless motor. In this example, only positive-phase outputs (WG1, WG2, WG3) are PWM pulses. This method is called an "Upper-arm Chopper." In this case mode 3 is selected for counter control. The value written to the timer reload buffer (WG_RELOAD) is equivalent to the PWM period and the counter works in the up-only mode. The phase compare buffers and the counter reload register are updated in the interrupt service routine. The output control buffer (WG_OUT) value is loaded into its register by the EPA software timer. The counter (WG_COUNT) is also set to 0001H at that time. Figure 7.10 shows the timing diagrams in this mode.

The last example is a 4 phase stepper motor driven by switched PWM, 1-2 phase excitation and a unipolar driver. The timings are shown in Figure 7.11.

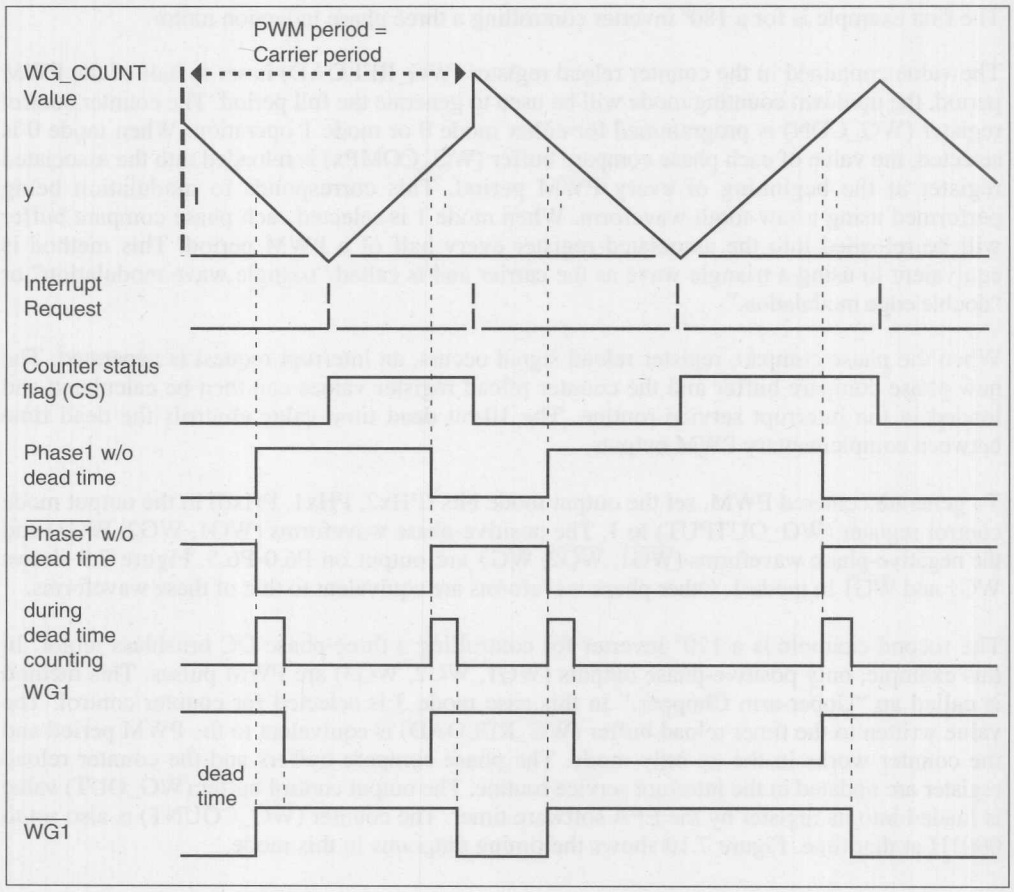


Figure 7.9. Phase1 Timing Diagram in Mode 1 with OP1=0 and OP0=0 (Active low for both WG1 and WG1)

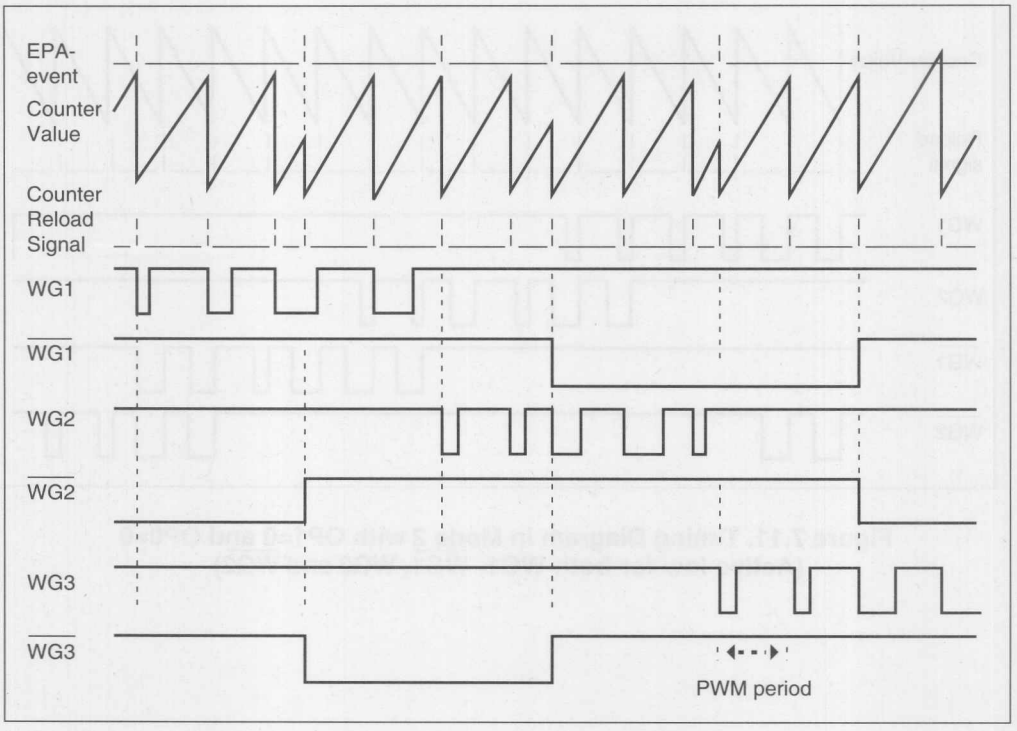


Figure 7.10. Timing Diagram in Mode 3 with OP1=0 and OP0=0 (Active low for WG1, WG1, WG2, WG2, WG3 and WG3)

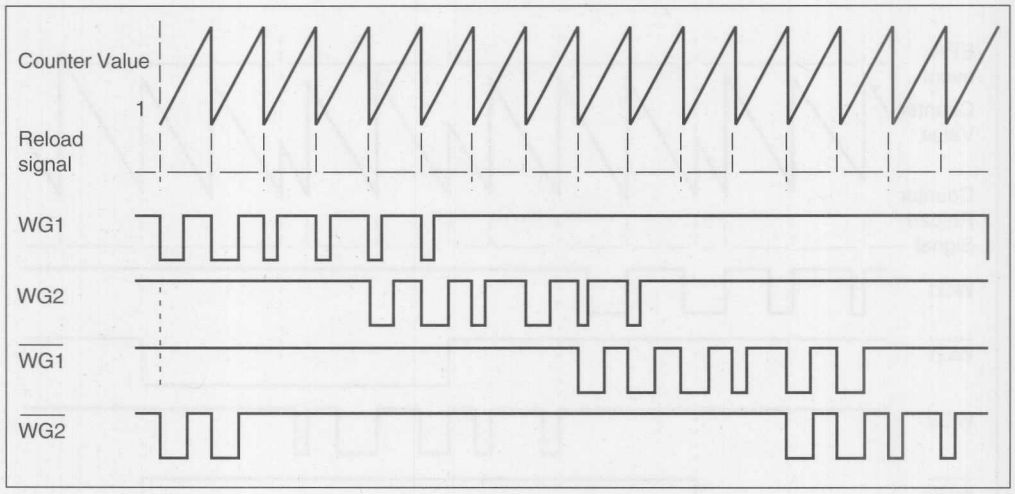


Figure 7.11. Timing Diagram in Mode 2 with OP1=0 and OP0=0 (Active low for both WG1, $\overline{WG1}$, WG2 and $\overline{WG2}$)

Pulse Width Modulation (PWM) 8

CHAPTER 8
PULSE WIDTH MODULATION

The 8XC196MC, in addition to the waveform generator, has two generic PWM modules. Each channel has a programmable duty cycle of 8-bits resolution and the period of both can be programmed through an 8-bit period control register. Other parts of the module are: an 8-bit counter, and two 8-bit PWM compare registers. The PWM output pins are shared with P6.6 and P6.7. The output control bits are in the output control register of waveform generator peripheral (WG_OUT).

The following pins are associated with the PWM:

P6.6	PWM0	(PWM Output 0)
P6.7	PWM1	(PWM Output1)

The following registers are associated with the PWM:

Address	Register	Description
1FB0H	PWM0_DUTY	PWM 0 Duty Cycle Control Register
1FB2H	PWM1_DUTY	PWM 1 Duty Cycle Control Register
1FB4H	PWM_PERIOD	PWM Period Control Register
1FB6H	PWM_PER_CNT	PWM Period Count Register
1FC0H	WG_OUT	Waveform Generator Output Control Register

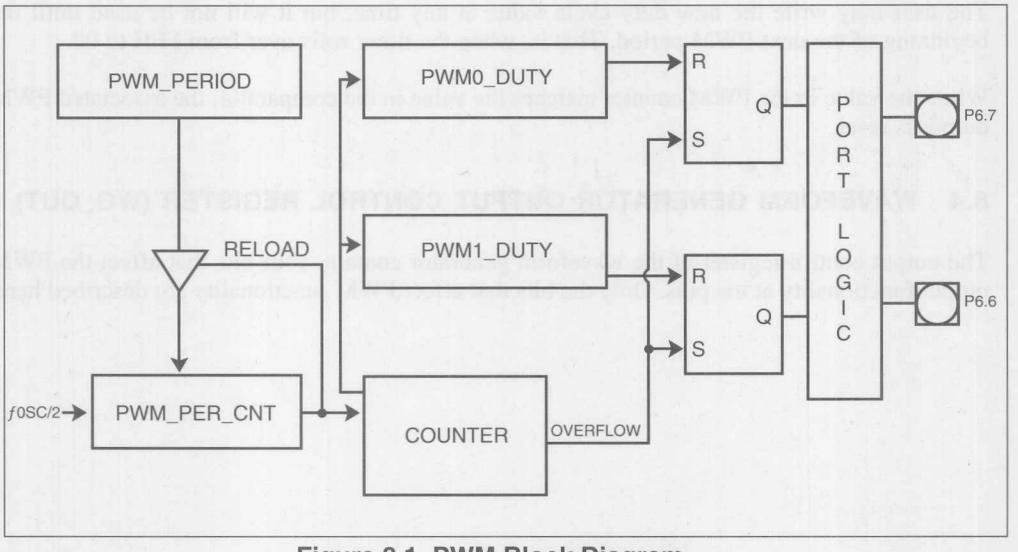


Figure 8.1. PWM Block Diagram

8.1. PWM PERIOD REGISTER (PWM_PERIOD)

This 8-bit register controls the period of the PWM outputs. This register can be read and written at location 1FB4H. It contains a value which determines the number of state counts necessary for incrementing the PWM counter. The formula for determining the PWM frequency is as follows:

$$\text{PWM Freqnency} = \frac{\text{fOSC}}{512\ 3\ (\text{PWM\_PERIOD Value} + 1)}$$

The contents of PWM_PERIOD is loaded into the PWM period count register whenever the count equals 0.

8.2. PWM PERIOD COUNT REGISTER (PWM_PER_CNT)

When read, this register provides the current value of the decremented period counter. This is provided to increase test coverage capability and decrease test time. This register cannot be written.

8.3 PWM0 AND PWM1 REGISTERS

Each PWM channel has an 8-bit register (PWMx_DUTY) that controls the duty cycle of the corresponding channel. A 0 loaded into this register will cause the PWM to output a low continuously (0% duty cycle). An FFH in this register will cause the PWM to have its maximum duty cycle (99.6% duty cycle). A continuously high PWM output may be obtained by using the waveform generator output control register (WG_OUT). See Section 7.1.5.

The user may write the new duty cycle value at any time, but it will not be used until the beginning of the next PWM period. That is, when the timer rolls over from FFH to 00.

When the value in the PWM counter matches the value in the comparator, the associated PWM output is reset.

8.4 WAVEFORM GENERATOR OUTPUT CONTROL REGISTER (WG_OUT)

The output control register of the waveform generator contains four bits that affect the PWM output functionality at the pins. Only the bits that affect PWM functionality are described here.

WG_OUT (1FC0H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP1	OP0	SYNC	PE1	PE0	PH32	PH22	PH12	P1	P0	PH31	PH30	PH21	PH20	PH11	PH10

PE1, PE0: Enable P6.7, P6.6 (respectively) as PWM output

P1, P0: P6.7, P6.6 (respectively) output value

PEn	Pn	Pin Output
0	0	0
0	1	1
1	X	PWM Output

Figure 8.2. WG Output Control Register (WG_OUT)

The placement of these bits allows all of Port 6 to be written using the lower byte of WG_OUT.

8.5 PWM COUNTER

This is an 8-bit counter that continually increments. The increment of the register is controlled by the PWM period circuitry described earlier. An overflow of this register sets the PWM outputs. This register cannot be read or written.

Analog-to-Digital (A/D) Converter (Ports 0 and 1)

9

CHAPTER 9
ANALOG-TO-DIGITAL (A/D) CONVERTER (PORTS 0 AND 1)

The analog-to-digital (A/D) converter handles analog inputs to the 8XC196MC. As Figure 9.1 illustrates, the A/D converter has a 13-channel multiplexer, a sample and hold and a 10-bit successive approximation A/D converter. The A/D converter inputs are shared with the Port 0 pins and Port 1 pins 0-4.

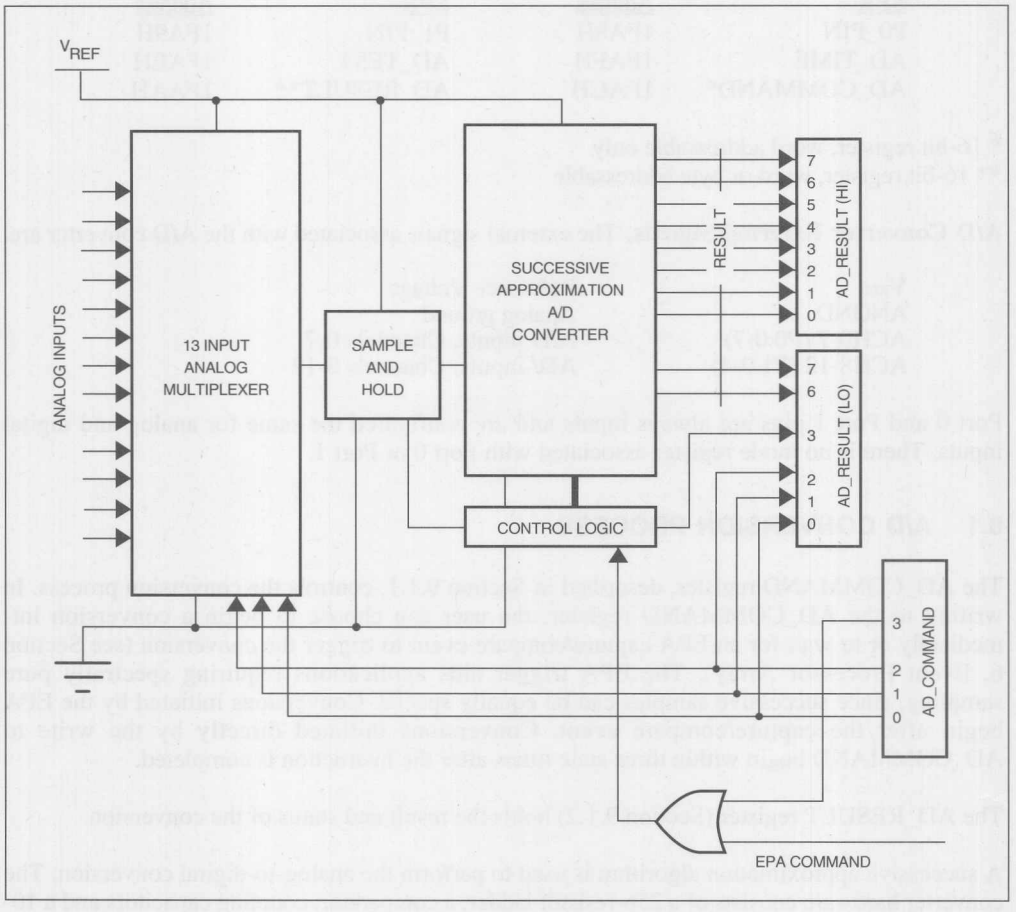


Figure 9.1. A/D Converter Block Diagram

The 8XC196MC's A/D converter is an improved version of the 80C196KC's A/D converter. The A/D can perform either 8- or 10-bit conversions. An 8-bit conversion trades resolution for

faster conversion time. Programmable sample and hold time and A/D conversion times are available as well as an 8-bit threshold detect mode, which sets an interrupt flag when the programmed voltage equals the input voltage. In addition, the A/D offers a number of test modes. Zero offset compensation circuitry allows programmable hardware offset adjustment.

Automated A/D conversions and result storage are facilitated by the A/D scan mode of the peripheral transaction server (PTS), Section 12.3.3.

A/D Converter SFRs. The A/D Converter SFRs are single-byte registers except as noted.

SFR	Address	SFR	Address
P0_PIN	1FA8H	P1_PIN	1FA9H
AD_TIME	1FAFH	AD_TEST	1FAEH
AD_COMMAND*	1FACH	AD_RESULT**	1FAAH

* 16-bit register, word addressable only
** 16-bit register, word or byte addressable

A/D Converter External Signals. The external signals associated with the A/D converter are:

V_{REF}	Reference Voltage
ANGND	Analog ground
ACH0-7 (P0.0-7)	A/D inputs, Channels 0-7
ACH8-12 (P1.0-4)	AD/ inputs, Channels 8-12

Port 0 and Port 1 pins are always inputs and are configured the same for analog and digital inputs. There is no mode register associated with Port 0 or Port 1.

9.1 A/D CONVERSION PROCESS

The AD_COMMAND register, described in Section 9.1.1, controls the conversion process. In writing to the AD_COMMAND register, the user can choose to begin a conversion immediately or to wait for an EPA capture/compare event to trigger the conversion (see Section 6, Event Processor Array). The EPA trigger aids applications requiring spectrally pure sampling, since successive samples can be equally spaced. Conversions initiated by the EPA begin after the capture/compare event. Conversions initiated directly by the write to AD_COMMAND begin within three state times after the instruction is completed.

The AD_RESULT register (Section 9.1.2) holds the result and status of the conversion.

A successive approximation algorithm is used to perform the analog-to-digital conversion. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic to control the process. The resistor ladder provides 20 mV steps (V_{REF} = 5.12V), while capacitive coupling creates 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltage levels are

available for comparison against the analog input to generate a 10-bit conversion result. 8-bit conversions use only the resistor ladder, providing 256 levels.

The successive approximation conversion compares a reference voltage to the analog input voltage stored in the sampling capacitor. A binary search is performed for the reference voltage that most closely matches the input. The 1/2 full scale reference voltage is the first tested. This corresponds to a 10-bit result in which the most significant bit is zero, and all other bits are one (0111.1111.11b). If the analog input is less than the test voltage, bit 10 is left at zero and a new test voltage of 1/4 full scale (0011.1111.11b) is tested. If this test voltage is less than the analog input voltage, bit 9 of the SAR is set and bit 8 is cleared for the next test (0101.1111.11b). This binary search continues until 8 or 10 tests have occurred, at which time the valid 8-bit or 10-bit result resides in the SAR where it can be read by software.

NOTE

AD_RESULT can be read before the conversion process is complete, but this is not recommended. Partial A/D conversions are NOT guaranteed to be accurate. Partial conversion times are calculated according to the rules in Section 9.2.

AD_COMMAND (1FACH)

7	6	5	4	3	2	1	0
0	TD	MODE	GO	A/D Channel			

A/D Channel	Channel number	
GO	0 = start by EPA 1 = start now	
MODE	For TD = 0:	0 = 10-bit conversion 1 = 8-bit conversion
	For TD = 1:	0 = Detect high 1 = Detect low
TD	(Threshold Detect) 0 = Normal Conversion 1 = Threshold Detection	
Bit 7	0	

Figure 9.2. AD_COMMAND Register

9.1.1 A/D Command Register (AD_COMMAND)

The AD_COMMAND register controls the conversion process according to the bit assignments in Figure 9.2. Bits 0-3 hold the channel to be monitored. The GO bit determines if the conversion is to be initiated by an EPA compare event(GO = 0) or immediately upon the write to AD_COMMAND (GO = 1). This register is buffered, so it is possible for the EPA to start one A/D conversion while another is in progress. If this happens, the conversion in

progress is aborted and the new one is started. The TD bit selects the A/D operation as a Normal Conversion (TD = 0) or a Threshold Detection (TD = 1).

Normal Conversion. In a Normal Conversion (TD = 0), the MODE bit determines the number of bits for the conversion. An 8-bit conversion provides increased conversion speed for users requiring only 8-bit resolution and faster conversion time. An 8-bit conversion requires less time than does a 10-bit conversion for two reasons: there are two fewer bits to resolve, and the comparator requires less settling time for 20 mV resolution than for 5 mV resolution. When the conversion is finished, the result is placed in AD_RESULT, and the A/D DONE interrupt flag is set.

Threshold Detection. When Threshold Detection (TD = 1) is used, the MODE bit selects Detect High (MODE = 0) or Detect Low (MODE = 1). To set the threshold value an 8-bit value is written to the high byte of AD_RESULT. This selects a reference voltage which is compared with an analog input pin. When the voltage on the analog input pin crosses over (Detect High) or under (Detect Low) the threshold value, the A/D DONE interrupt flag is set.

NOTE

While the threshold detect mode is selected for an analog input pin, no other conversion can be started. If another value is loaded into AD_COMMAND, the threshold detect mode is halted and the new command is executed.

9.1.2 A/D Result Register (AD_RESULT)

The AD_RESULT register (Figure 9.3) contains A/D status information and either the most recently completed conversion result (for a normal conversion) or the threshold value (for a threshold detection). For a 10-bit conversion, the result is stored in the 10 most significant bits of the word. For an 8-bit conversion, the lower 2 of these 10 bits are undefined. Bits 0-3 show which channel was converted/detected. The busy bit indicates whether or not the A/D is currently working on a conversion/detection.

For a normal conversion, results are interpreted using one of the following formulas.

$$\text{RESULT (8-bit)} = 255 * (A_{IN} - \text{ANGND}) / (V_{REF} - \text{ANGND})$$

$$\text{RESULT (10-bit)} = 1023 * (A_{IN} - \text{ANGND}) / (V_{REF} - \text{ANGND})$$

where A_{IN} is the analog input voltage.

9.2 A/D TIME REGISTER (AD_TIME)

The AD_TIME register (Figure 9.4) allows the user to program an optimal conversion speed for the resolution and frequency being used. Two parameters (SAM and CONV) control the amount of time used for an A/D conversion. The total number of states used is:

A/D States = Sample States + Conversion States

where:

Sample States =	4 3 (SAM) + 1
Conversion States =	B 3 (CONV + 1) + 1.5
B =	number of bits for conversion (8 or 10)
SAM =	Value in AD_TIME BITS 5,6,7; must be 1-7
CONV =	Value in AD_TIME BITS 0-4; must be 2-31

The total time for A/D conversion is:

T = Sample Time + Conversion Time

= State Time 3 Sample States + State Time 3 Conversion States

= State Time 3 (4 3 SAM + 1) + State Time 3 (B 3 (CONV + 1) + 1.5)

= State Time 3 ([4 3 SAM]+ [B 3 (CONV + 1)]+ 2.5)

The upper three bits of the AD_TIME register (SAM:bits 5–7) control the sample window time, while the lower five bits (CONV: bits 0–4) control the conversion time per bit. Figure 9.4 shows the solutions for SAM and CONV, given the state time, the sample time, and the bit conversion time.

AD_RESULT (1FAAH)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8 MSB								2 LSB		RSV	BUSY	A/D Channel			

A/D Channel	Channel number: 0-12
BUSY	0 = A/D idle 1 = A/D in use
RSV	Reserved, write 0
2 LSB	2 least significant bits of A/D Result
8 MSB	8 most significant bits of A/D Result

Figure 9.3. AD_RESULT Register

Figure 9.5 shows an example time chart. The conversion time ranges represent speeds at which the A/D converter could be run, not the speeds for accurate conversion.

9.2.1 Considerations for Setting SAM and CONV

The sample time (determined by SAM) is the length of time that the analog input voltage is actually connected to the sample capacitor. If this time is too short, the sample capacitor will not charge completely. If the sample time is too long, the input voltage may change and cause conversion errors. The convert time (determined by CONV) is the length of time required to convert one bit of the analog input voltage stored on the sample capacitor to a digital value. The convert time must be long enough for the comparator and circuitry to settle and resolve the voltage. Excessively long conversion times allow the sample capacitor to discharge, degrading accuracy.

AD_TIME (1FAFH)

7	6	5	4	3	2	1	0
SAM			CONV				

SAM = ((STIME * Fosc/2) - 1) / 4

CONV = (((CTIME * Fosc/2) -1.5) / B) - 1

SAM = 1 to 7
CONV = 2 to 31
STIME = Sample time, µS
CTIME = Convert time, µS
Fosc = XTAL1 frequency, MHz
B = 8 or 10 (# bits in conversion)

Figure 9.4. AD_TIME Register

Freq.	State Time	Range of Sample Window Time
4 MHz	500 ns	2.5 µs–14.5 µs
8 MHz	250 ns	1.25 µs–7.25 µs
12 MHz	166 ns	0.83 µs–4.81 µs
16 MHz	125 ns	0.625 µs–3.625 µs

		Range of Conversion Time	
Freq.	State Time	8 Bits	10 Bits
4 MHz	500 ns	12.8 µs–129 µs	15.8 µs–161 µs
8 MHz	250 ns	6.37 µs–64.4 µs	7.88 µs–80.4 µs
12 MHz	166 ns	4.23 µs–42.7 µs	5.23 µs–53.4 µs
16 MHz	125 ns	1.39 µs–32.2 µs	3.93 µs–40.2 µs

See data sheet for recommended sample and conversion times.

Figure 9.5. Example Time Chart

The AD_TIME register programs only the speed at which the conversions are performed, **not** the speed at which it can convert correctly. Programming some speeds will cause erroneous A/D results. Refer to the data sheet for recommended sample and conversion times. Use the equations shown in Figure 9.4 to solve for SAM and CONV to place in the AD_TIME register.

Restrictions on AD_TIME

1. Since the AD_TIME is reset to 0FFH, this register must be initialized before A/D conversions are performed.
2. Initialize the AD_TIME register first, then initialize the AD_COMMAND register.
3. SAM must be ≥ 1.
4. CONV must be ≥ 2.

9.3 A/D TEST REGISTER (AD_TEST)

The AD_TEST register (Figure 9.6) specifies adjustments for DC offset errors. The offset voltage applies to all input channels. Bits 0, 1 and 4-7 are reserved and should be written as zeros.

AD_TEST (1FAEH)

7	6	5	4	3	2	1	0
0	0	0	0	Offset		0	0

Offset 00 = no change
01 = add 2.5 mV
10 = subtract 2.5 mV
11 = subtract 5.0 mV

Note: Offset range is based on V_{REF} = 5.12 V.

Bits 4 -7 0

Figure 9.6. AD_TEST Register

The offset adjustment selection allows the user to set the zero offset point (no adjustment, +2.5 mV, −2.5 mV, or −5.0 mV). This feature can eliminate or reduce off-chip compensation hardware.

9.4 A/D PORT STRUCTURE AND INTERFACE CIRCUITRY

9.4.1 A/D Port Structure

Port 0 and port 1 are used for the analog inputs to the A/D. Figure 9.7 shows the structure. The CPU can also read the digital values of the pins. Because the external inputs to the pins

are analog signals, the input buffers are usually powered down and activated only when the port is read. Port 0 and port 1 pins will **not** function as outputs.

The port 0 and port 1 pins are powered by V_{REF} and ANGND. For the port pins to function as either analog or digital inputs, the analog reference voltage (V_{REF}) and analog ground (ANGND) **must** be connected.

Port 0 and port 1 pins are special in that they can be used individually as digital inputs and analog inputs at the same time. When used as digital inputs, the pins act as high-impedance inputs. When used as analog inputs, however, they are required to provide current to the internal sample capacitor when a conversion begins. The input capacitance of a pin increases when it is being sampled. When a digital input is applied, the pin is sampled only when the register Px_PIN is read. The data must be stable one state time before the read takes place. It is not recommended to read either port during (analog) sampling, as this can introduce noise on the channel and decrease the conversion accuracy.

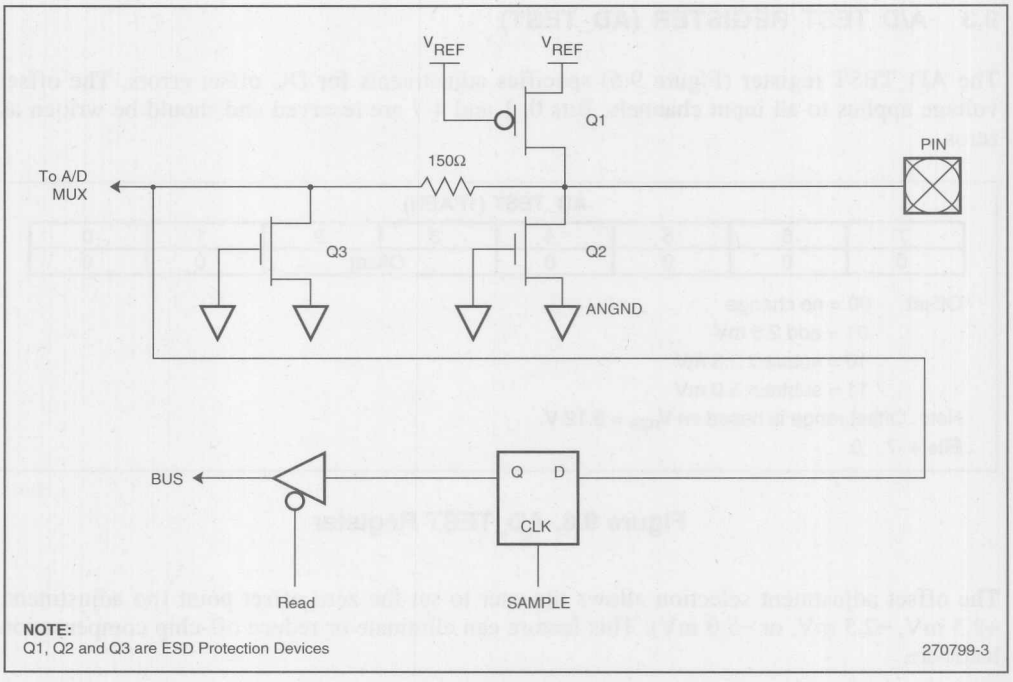


Figure 9.7. Port 0 and Port 1 Structure

9.4.2 A/D External Interface Circuitry

The external interface circuitry to an analog input is highly dependent upon the application and can impact converter characteristics. Several important factors must be considered in the

external interface design: input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor. These factors are idealized in Figure 9.8. The following example is for a 1 μs sample time and a 10-bit conversion. Consult the latest A/D specifications for the optimum sample time. The external input circuit must be able to charge a sample capacitor (C_S) through a series resistance (R_I) to an accurate voltage, given a DC leakage (I_L). Typically, C_S is around 2 pF, R_I is around 1.2 KΩ, and I_L is approximately 1 μA. Consult the latest data sheet for guaranteed specifications.

External circuits with source impedances of 1 KΩ or less can maintain an input voltage within a tolerance of about 0.2 LSB (1.0 KΩ × 1.0 μA = 1.0 mV) given the DC leakage. Source impedances above 5 KΩ can result in an external error of at least one LSB due to the voltage drop caused by the 1 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy because the internal sample capacitor will not charge completely during the sample time.

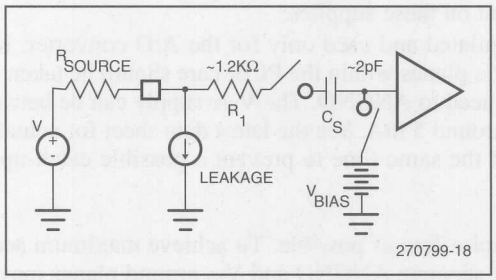


Figure 9.8. Idealized A/D Sampling Circuitry

Source impedance requirements relax if an external capacitor of sufficient size is attached directly to the analog input pin. Since the internal sample capacitor is around 2.0 pF, an external 0.005 μF capacitor (2048 × 2.0 pF) should provide an accurate input voltage to 0.5 LSB. If there is leakage on the capacitor, this leakage must be added to the pin leakage discussed above.

Placing an external capacitor on each analog input also reduces the sensitivity to noise because the capacitor combines with series resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to the external capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor also limits input current during over-voltage conditions.

Figure 9.9 shows a simple analog interface circuit based on the discussion above. The circuit in the figure also provides limited protection against over voltage and under voltage conditions on the analog input. Should the input voltage drop significantly below ground, diode D2 will forward bias at about 0.8 DCV. This limits the current sourced by the input pin to a few milliamps. If the external signal can supply more than 100 mA, a 100-to-200 ohm resistor between the source and the protection circuit may be desirable. However, before any circuit is used, it should be thoroughly analyzed for the specific application.

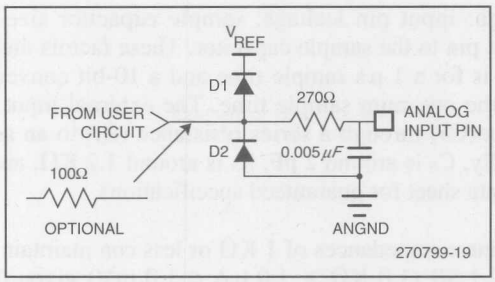


Figure 9.9. Suggested A/D Input Circuit

Analog References. Reference supply levels strongly influence the absolute accuracy of the A/D conversion. Bypass capacitors should be used between V_{REF} and ANGND in order to reduce noise that may be present on these supplies.

V_{REF} should be well regulated and used only for the A/D converter. In designs implementing separate ANGND and V_{SS} planes within the PCB, care should be taken to ensure that the device regulating V_{REF} is referenced to ANGND. The V_{REF} supply can be between 4.5 V and 5.5 V and must be able to source around 5 mA. See the latest data sheet for actual specifications. V_{REF} and V_{CC} should power-up at the same time to prevent a possible catch-up condition occurring on V_{REF}.

ANGND should be as noise-free as possible. To achieve maximum accuracy and performance from the A/D converter, separate ANGND and V_{SS} ground planes may be implemented within the PCB. Common practice is to implement these ground planes on the same plane within the PCB, and to separate them with a single non-conductive perimeter. If this is implemented, these two ground planes should be connected at one point close to the microcontroller. The purpose of separating ANGND and V_{SS} is to prevent excessive digital noise that may exist on a PCB from corrupting the A/D conversion. Regardless of how grounding is implemented, it is important to keep ANGND as close to V_{SS} as possible, preferably within 50 mV. If the difference between ANGD and V_{SS} exceeds 0.4V, the device may catch-up.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC}. However, this is a more noisy configuration than separately regulated voltage sources. In addition, V_{REF} and ANGND must be connected, even if the A/D converter is not being used. The port 0 and port 1 pins receive their power from the V_{REF} and ANGND pins, even when they are used for digital input.

9.5 THE A/D TRANSFER FUNCTION

The conversion result is a 10-bit ratiometric representation of the input voltage. The numerical value obtained from the conversion is:

RESULT (10-Bit) = 1023 * (A_{IN} - ANGND) / (V_{REF} - ANGND)

This produces a stair-stepped transfer function when the output code is plotted versus input voltage (see Figure 9.10). The resulting digital codes can be taken as simple ratiometric information, or they can provide information about absolute voltages on the inputs.

The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, closing a servo-loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error, zero offset, full-scale error, differential non-linearity and non-linearity. These are ''transfer function'' errors related to the A/D converter. In addition, converter temperature drift, V_{CC} rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching and random noise should be considered. Fortunately, an ''absolute error'' specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. However, the various sub-components of error are important in many applications. These error components are described in Section 9.6 and in the text below where ideal and actual converters are compared.

An unavoidable error results simply from the conversion of a continuous voltage to its integer digital representation. This error is called quantizing error, and is always 0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 9.10 shows the characteristic for an ideal 3-bit A/D converter (i.e., the ideal characteristic).

Note that in Figure 9.10 the ideal characteristic possesses unique features: its first code transition occurs when the input voltage is 0.5 LSB; its full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and its code widths are all exactly one LSB. These characteristics result in a digitization without offset, full-scale or linearity errors. In other words, this is a perfect conversion.

Figure 9.11 shows an actual characteristic of a hypothetical 3-bit converter that is imperfect. When the ideal characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called ''zero offset,'' and the deviation of the final code transition from ideal is ''full-scale error.'' The deviation of the code widths from ideal causes two types of errors: differential non-linearity and non-linearity. Differential non-linearity is a local linearity error measurement, whereas non-linearity is an over-all linearity error measure.

Differential non-linearity is the degree to which actual code widths differ from the ideal one LSB width. It gives the user a measure of how much the input voltage may have changed in order to produce a one-count change in the conversion result. Non-linearity is the worst case deviation of code transitions from the corresponding code transitions of the ideal characteristic. Non-linearity describes how the extent to which differential non-linearities can add up to produce an overall maximum departure from an ideal characteristic. If the differential non-linearity errors are too large, it is possible for an A/D converter to miss codes or exhibit non-

monotonicity. Neither behavior is desirable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction.

Differential non-linearity and non-linearity are quantified by measuring the terminal based linearity errors. A terminal based characteristic results when an actual characteristic is shifted and rotated about zero to eliminate zero offset and full-scale error (see Figure 9.12). The terminal based characteristic is similar to the actual characteristic that would be seen if zero offset and full-scale errors were externally trimmed away. In practice, this is done by using input circuits that include gain and offset trimming. In addition, V_{REF} can also be closely regulated and trimmed to affect full-scale error adjustment.

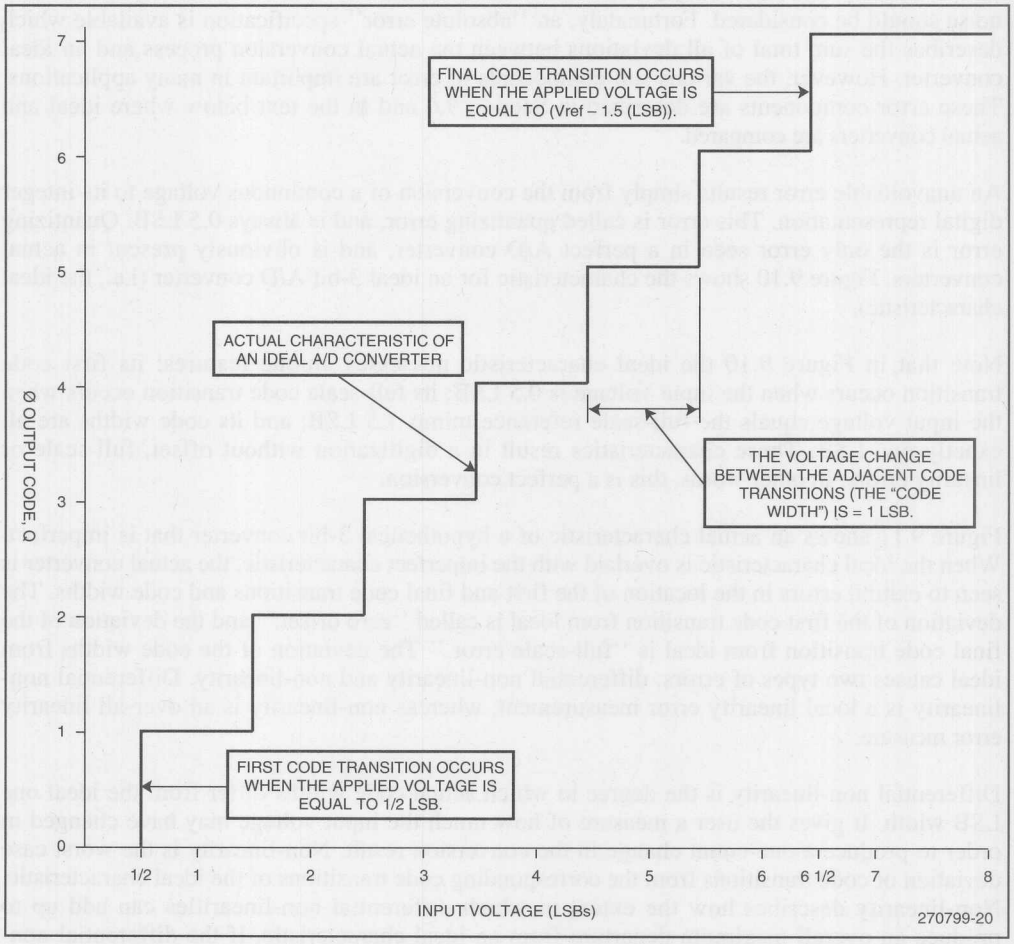


Figure 9.10. Ideal A/D Characteristic

Other factors that affect a real A/D converter system include sensitivity to temperature, failure to completely reject all unwanted signals, multiplexer channel dissimilarities and random noise. Fortunately these effects are small.

Temperature sensitivities are described by the rate at which typical specifications change with a change in temperature.

Undesired signals come from three main sources:

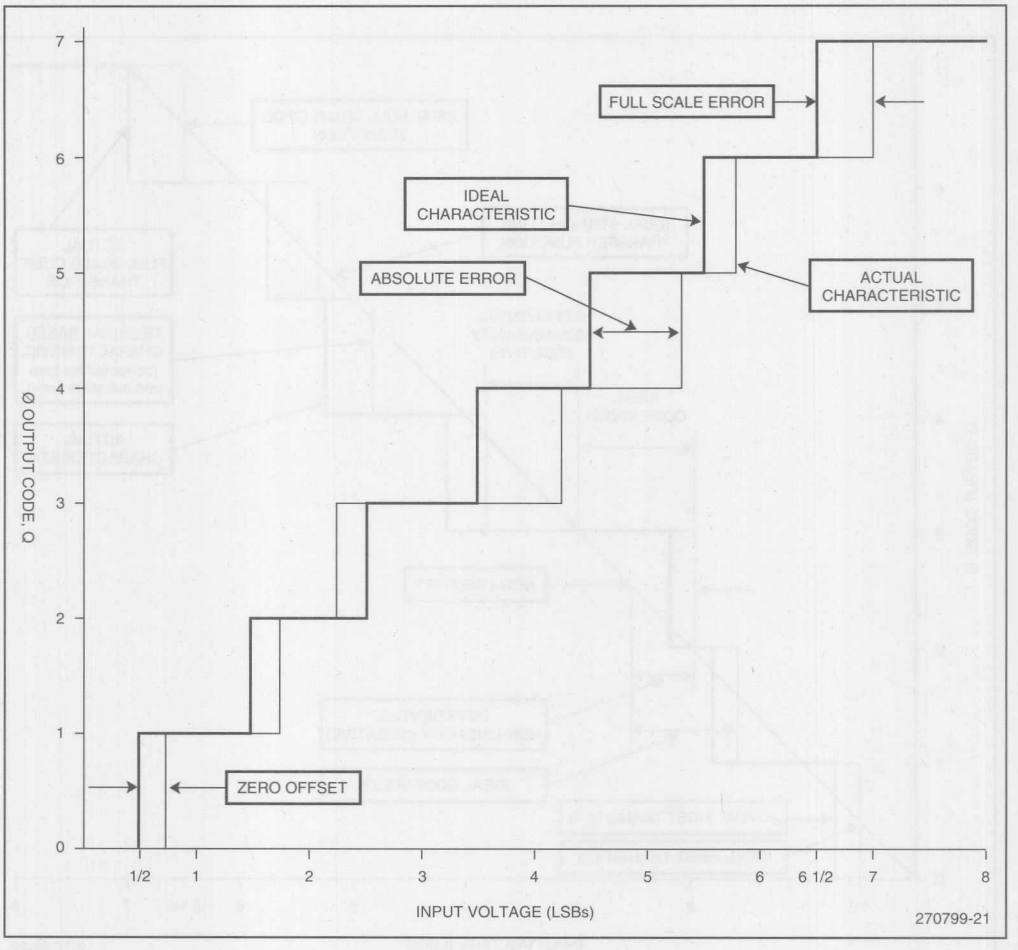


Figure 9.11. Actual and Ideal A/D Characteristic

- Noise on V_{CC} — V_{CC} rejection.
- Input signal changes on the channel being converted after the sample window has closed — Feedthrough.
- Signals applied to channels not selected by the multiplexer — off-isolation.

Finally, multiplexer on-channel resistances differ slightly from one channel to the next, causing channel-to-channel matching errors, and random noise in general results in repeatability errors.

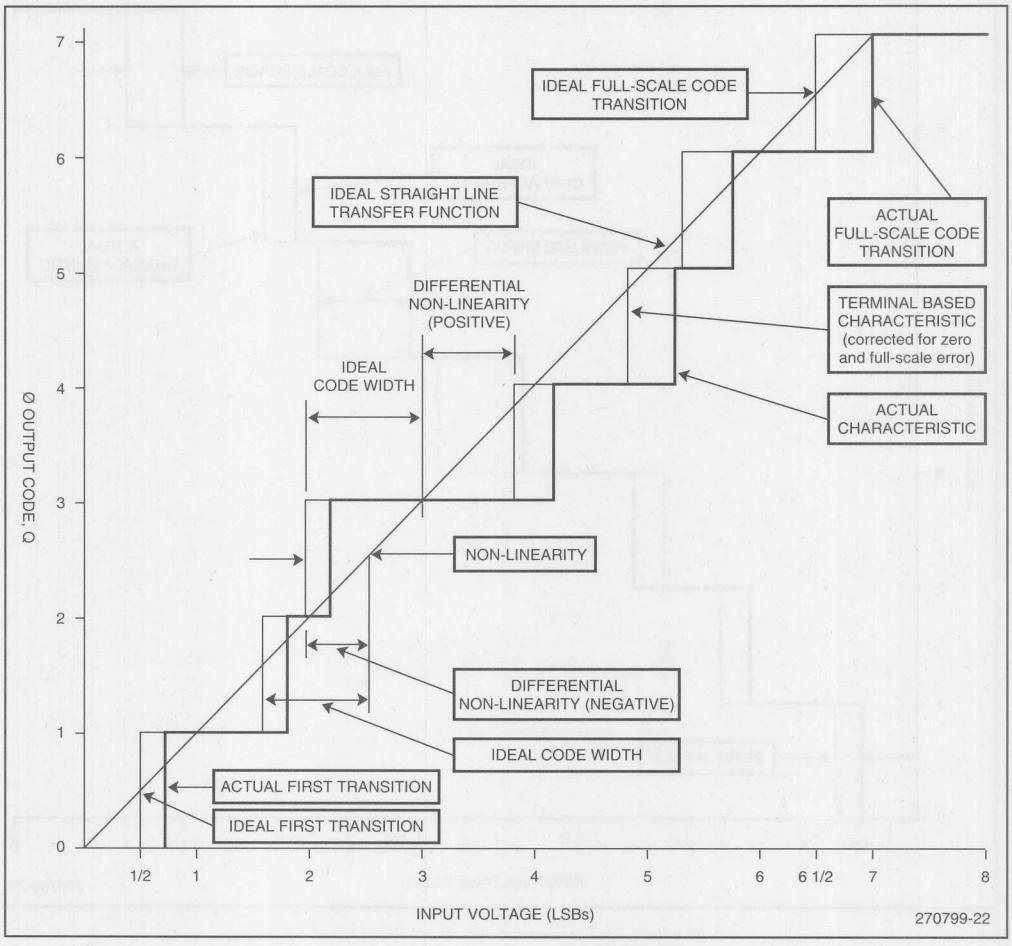


Figure 9.12. Terminal Based A/D Characteristic

9.6 A/D GLOSSARY OF TERMS

Figures 9.10, 9.11 and 9.12 display many of these terms. Refer to AP-Note AP-406 "MCS-96 Analog Acquisition Primer" for additional information on the A/D terms.

ABSOLUTE ERROR. The maximum difference between corresponding actual and ideal code transitions. Absolute error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC. The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage and frequency conditions. An actual characteristic rarely has ideal first and last transition locations of ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE. The multiplexer property which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g., the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING. The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC. A graph of input voltage versus the resultant output code from an A/D converter. It describes the transfer function of the A/D converter.

CODE. The digital value output by the converter.

CODE CENTER. The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION. The point at which the converter changes from an output code of Q, to a code of Q+1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH. The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK. See "Off-Isolation."

DC INPUT LEAKAGE. Leakage current to ground from an analog input pin during a conversion.

DIFFERENTIAL NON-LINEARITY. The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH. Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL SCALE ERROR. The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC. A characteristic with its first code transition at VIN = 0.5 LSB, its last code transition at VIN = (V_{REF} - 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE. The effective series resistance from the analog input pin to the sample capacitor.

LSB (LEAST SIGNIFICANT BIT). The voltage value corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12V, one LSB is 5.0 mV. For an 8-bit conversion, one LSB equals 20 mV. Note that this is different from digital LSBs, because an uncertainty of two LSBs, when referring to a 10-bit A/D converter, equals 10 mV.

MONOTONIC. The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES. For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY. The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristics.

OFF-ISOLATION. Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as crosstalk.)

REPEATABILITY. The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION. The number of input voltage levels that the converter can unambiguously distinguish between. This also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY. The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY. The variation in the sample delay.

SAMPLE TIME. The time that the sample window is open.

SAMPLE TIME UNCERTAINTY. The variation in the sample time.

SAMPLE WINDOW. The sample window begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION. An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS. The change in the stated variable per degree Centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effects of temperature drift.

TERMINAL BASED CHARACTERISTIC. An actual characteristic which has been rotated and translated to remove a zero offset and full-scale error.

VCC REJECTION. Ratio of the change in the A/D characteristic to the change in V_{CC}.

ZERO OFFSET. The difference between the expected and actual input voltage corresponding to the first code transition.

Memory Mapped I/O Ports 3,4 and 5

10

Memory Mapped I/O
Ports 3,4 and 5

CHAPTER 10
MEMORY MAPPED I/O PORTS 3, 4 AND 5

Ports 3, 4 and 5 are memory mapped I/O ports. Memory mapped I/O can be addressed only via 16-bit addresses. Windowing is not possible.

These ports have functions in addition to memory mapped I/O. Ports 3 and 4 constitute the address/data bus (Section 15). The port 5 pins carry bus control signals.

Addressing Range for Memory Mapped Ports. The memory mapped ports and registers reside in the address range 1FE0H-1FFFH. Data references to ports 3 and 4 are directed externally if $\overline{EA}$ = 0 or internally if $\overline{EA}$ = 1. All data references to port 5 are always directed internally. This means that port 5 cannot be reconstructed.

Memory Mapped I/O Port Registers. The register locations for ports 3, 4 and 5 are shown below.

Register	Address	Register	Address
P4_PIN	1FFFH	P5_PIN	1FF7H
P3_PIN	1FFEH	P5_REG	1FF5H
P4_REG	1FFDH	P5_DIR	1FF3H
P3_REG	1FFCH	P5_MODE	1FF1H

Memory Mapped I/O External Signals. The port pins serve as memory mapped I/O and for the following special functions.

P3.0-7 External System Bus (AD0-7) and Slave Port

P4.0-7 External System Bus (AD8-15)

P5.0-7 Special function signals:

P5.0	ALE	Address latch enable
P5.1	INST	Instruction fetch from external memory
P5.2	$\overline{WR}$	External memory write
P5.3	$\overline{RD}$	External memory read
P5.4		
P5.5	$\overline{BHE}$	Byte high enable
P5.6	READY	Ready
P5.7	BUSWIDTH	Bus width

10.1 PORTS 3 AND 4

Ports 3 and 4 provide memory mapped I/O or serve as the external address/data bus. Port 3 also serves as the slave port. The next section discusses the port circuit operation, and a following section describes how to use the ports.

10.1.1 Circuit Operation

Figure 10.1 is a schematic of Port 3 and 4 and Figure 10.2 shows the logic table. During reset (Section 14.5) the $\overline{\text{RESET}}$ signal turns off Q_U and Q_L and turns on the pull-up transistor (Q_W) to establish a weak logical 1. Q_W can source at least 10 µA at V_{CC} − 1V. During normal operation the port is controlled by FCN SELECT, which is not directly available to the user. If the 8XC196MC requires access to external memory, it assigns FCN SELECT = 0 to select the ADDR/DATA input to the multiplexer. ADDR/DATA then drives Q_U and Q_L as a complementary output. Q_U can source at least −3mA at V_{CC} − 0.7V and Q_L can sink at least 3mA at 0.45V. If external memory access is not required, the 8XC196MC sets FCN SELECT = 1 to select Px_REG as the input to the multiplexer. Px_REG, which latches PORT DATA, then drives Q_U and Q_L as an open drain output, which requires an external pull-up resistor. With PORT DATA = 1, the pin can be used as input. The signal on the port pin is latched in the register Px_PIN.

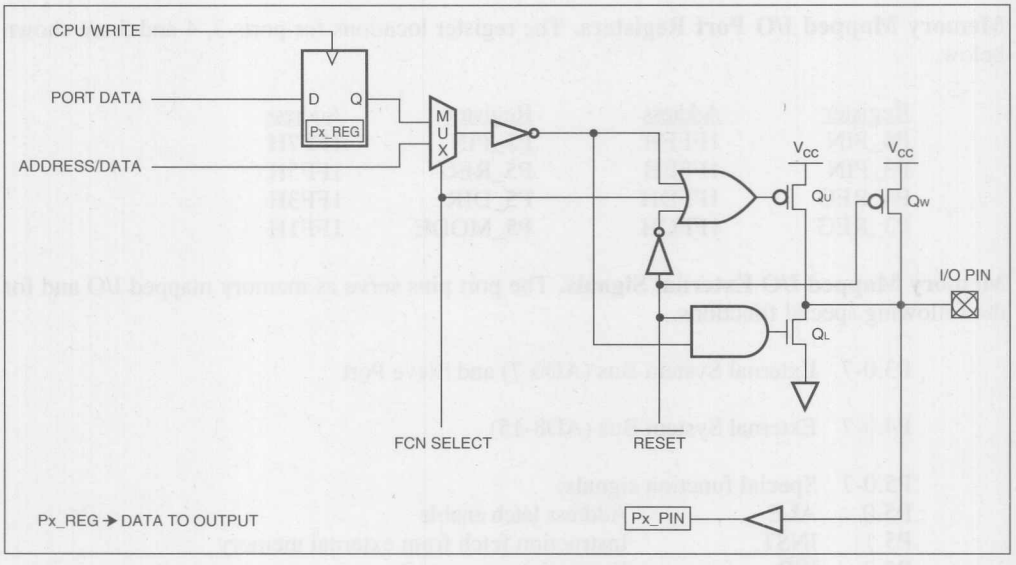


Figure 10.1. Circuit Schematic for Ports 3 and 4

10.1.2 Using Ports 3 and 4

External Address/Data Bus. The 8XC196MC automatically uses ports 3 and 4 for the external address/data bus whenever it requires access to external memory. Port 3 carries the lower byte of the address or data, and port 4 carries the upper byte. The port signals can be read from the Px_PIN register. Section 15 describes the operation of the external address/data bus.

Memory Mapped I/O. To use a port pin for output, write the output data to the port register Px_REG. These are open drain ports, so it may be necessary to connect an external pull-up

resistor. Note that whenever the 8XC196MC requires access to external memory, it will take over the port and drive the address/data bit onto the pin. Thus, the user's output is replaced during this time. When the external access mode is deactivated, the data written by the user to Px_REG is restored to the output pins. The state of the pins, whether determined by the user's write to Px_REG or the address/data, can be checked by reading Px_PIN.

To use a port pin for input, write a 1 to that bit in Px_REG (to drive the output to a high-impedance state) and read the input value on the bit in Px_PIN. The 8XC196MC will take over the port whenever it requires external memory access. The input source must be configured to avoid contention on the bus.

FCN SELECT	0	0	0	0	1	1
PORT DATA	0	1	0	1	0	1
ADDR/DATA	0	0	1	1	X	X
Px_REG	0	1	0	1	0	1
Q_U	off	off	on	on	off	off
Q_L	on	on	off	off	on	off
Px_PIN	0	0	1	1	0	HZ
Port Config.	Complementary				Open-Drain	
Port Function	External Bus Access				I/O or Slave	

X = Don't Care
HZ = High Impedance

Figure 10.2. Port 3 and 4 Logic Table

10.2 PORT 5

The pins of port 5 can be individually configured to operate in a standard I/O mode or a special function mode, in which the pin carries the signal listed in Section 10.0. If a particular special function signal is not utilized in a application, the associated pin can be used for standard I/O, except for P5.4 that should not be configured as input. Thus, each pin can be configured to operate in one of four modes:

1. As a pin for a special function signal controlled by the 8XC196MC or an off-chip component.
2. As a standard complementary output pin.
3. As a standard output pin with open drain, requiring an external pull-up resistor.
4. As a standard, high-impedance input pin.

With respect to operating modes, port 5 is similar to ports 2 of Section 5. Port 5 is different, however, in that the registers must be accessed using 16-bit addressing and cannot be windowed.

Section 10.2.1 discusses the basic operation of the port circuits and registers. Sections 10.2.2 and 10.2.3 describe how to configure the port pins for the standard I/O modes and for the special functions. Notes on the use of particular port pins and their associated special function signals are in Section 10.2.4

10.2.1 Circuits and Registers for Port 5

The circuit schematic for a single pin of Port 5 is shown in Figure 10.3. Signals $\overline{\text{PPU}}$ and $\overline{\text{WKPU}}$ are activated during reset (described in Section 14.5) to initialize the port pins at a weak 1. $\overline{\text{PPU}}$ is forced low for about 300 ns by the falling edge of $\overline{\text{RESET}}$ and pulls the pin high. The active low level of $\overline{\text{RESET}}$ then forces $\overline{\text{WKPU}}$ low, which weakly holds the pin high (at least 10 μA at V_{CC} - 1V). $\overline{\text{WKPU}}$ remains high and the pin remains at logical 1 until the user configures the port pin by writing to the register that selects the port (register P5_MODE described below).

P5_MODE, Port 5 mode register. Each bit in P5_MODE determines whether the corresponding pin functions as a standard I/O port pin (bit = 0) or is used for a special function signal (bit = 1).

P5_DIR, Port 5 I/O direction register. Each bit of P5_DIR specifies whether the corresponding pin functions as a complementary output (bit = 0) or as an input or open drain output (bit = 1). Pin P5.4 should not be configured as input, otherwise entry into test modes may occur.

P5_REG, Port 5 data output register. P5_REG contains data to be driven out by the respective pins. If the software writes data to P5_REG while P5_MODE = 0 and P5_DIR = 0, the effects are seen immediately on the pins. With P5_MODE = 1, the software or associated external component controls the pin. The software can still write to P5_REG, but the pin is unaffected until the corresponding P5_MODE bit is changed to 0. Thus, software can switch P5_MODE to 0, initialize or overwrite the port bit value, and change the P5_MODE bit back to 1. This feature can be used in initialization, fault recovery, exception handling, etc. without having to change the operation of the special function source.

P5_PIN, Port 5 pin input register. P5_PIN can be read to determine the current state of the pin, whether the pin is configured for the special signal or for standard output. The pin values are latched on phase 2 and transferred into the data register of the bus controller on phase 1 (see Section 14.4.2). As an input, P5.4 is used to enter factory test modes.

SF5_DIR, Port 5 special function I/O register. Port 5 has an additional register SF5_DIR, which is unavailable to the user. When the special function is selected (P5_MODE = 1), SF5_DIR determines whether the special function signal is an output (bit = 0) or an input (bit = 1). When the special function is selected for a pin, the appropriate bit of SF5_DIR is automatically written.

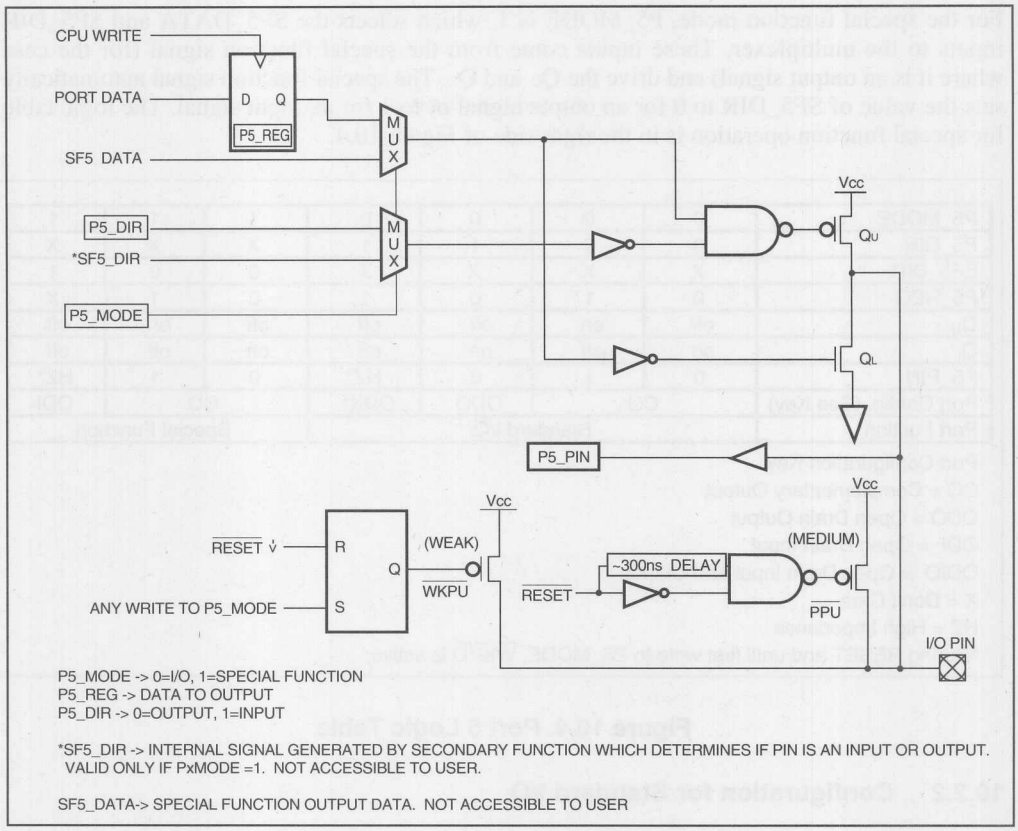


Figure 10.3. Circuit Schematic for PORT 5

The upper portion of Figure 10.3 shows the logic for driving the output transistor pair Q_U and Q_L. Q_U can source at least -3mA at V_{CC} - 0.7V and Q_L can sink at least 3mA at 0.45V. Circuit operation in the standard I/O modes and in the special function mode is discussed here. Subsequent sections give instructions for configuring the port pins for the desired modes.

For the standard I/O modes, P5_MODE is 0, which selects the PORT_DATA input to the multiplexer. When the user writes to P5_REG, the effect is to drive the PORT_DATA input, which drives the P5_REG flip-flop. In the standard I/O mode, the signal from SF5_DIR is blocked by the NAND gate. The P5_DIR and P5_REG signals are combined to drive the gates of Q_U and Q_L so that the output P5_PIN is high, low or high impedance (HZ). A logic table for standard I/O operation is given in the left side of Figure 10.4. The input mode is the same as the open drain mode with P5_PIN at high impedance (HZ), but an external pull-up is not required.

For the special function mode, P5_MODE is 1, which selects the SF5_DATA and SF5_DIR inputs to the multiplexer. These inputs come from the special function signal (for the case where it is an output signal) and drive the Q_U and Q_L. The special function signal automatically sets the value of SF5_DIR to 0 for an output signal or to 1 for an input signal. The logic table for special function operation is in the right side of Figure 10.4.

P5_MODE	0	0	0	0	1	1	1
P5_DIR	0	0	1	1	X	X	X
SF5_DIR	X	X	X	X	0	0	1
P5_REG	0	1	0	1	0	1	X
Q_U	off	on	off	off	off	on	off
Q_l	on	off	on	off	on	off	off
P5_PIN	0	1	0	HZ*	0	1	HZ*
Port Config. (See Key)	CO		ODO	ODIO	CO		ODI
Port Fuction	Standard I/O				Special Function		

Port Configuration Key:
CO = Complementary Output
ODO = Open Drain Output
ODI = Open Drain Input
ODIO = Open Drain Input and Output
X = Don't Care
HZ = High Impedance
*During RESET and until first write to P5_MODE, $\overline{\text{WKPU}}$ is active.

Figure 10.4. Port 5 Logic Table

10.2.2 Configuration for Standard I/O

To configure a port pin for standard I/O, write to the appropriate bits of P5_REG and P5_DIR, and then clear the corresponding bit of P5_MODE. Figure 10.5, based on the logic table of Figure 10.4, shows the bit values required for a desired configuration. For input mode, the register settings are the same as for open drain output at high impedance, but an external pull-up is not required.

I/O Mode	P5_PIN	P5_DIR	P5_REG	QU	QL
Output	0	0	0	off	on
(Complementary)	1	0	1	on	off
Output	0	1	0	off	on
(Open drain)	HZ*	1	1	off	off
Input**	HZ*	1	1	off	off

Notes: P5_MODE bit = 0
HZ = high impedence
*During RESET and until first write to P5_MODE $\overline{\text{WKPU}}$ is active
**Do not use P5.4 as an input.

Figure 10.5. Register Settings for Standard I/O Port Operation

PIN5.4 P5.4 stays at a weak 1 until a write to P5_MODE is performed. This pin should **not** be used as an input pin. If this pin is driven low during the low to high transition of $\overline{\text{RESET}}$, test modes could be entered. This would inhibit normal operation.

10.2.3 Configuration for Special Functions

To avoid transients during the transition from I/O mode to special function mode, you should write to the port registers in the following order:

1. Write to P5_REG
2. Write to P5_DIR
3. Write to P5_MODE

The following guidelines should be used to determine the value written to P5_REG.

- If the special function is an input, write a 1 to P5_REG to attain a high-impedance (input) configuration.
- If the special function is an output, write to P5_REG the expected value of the special function (usually the inactive value).

The value of P5_DIR is determined by the nature of the special function:

- = 0 for a complementary output (without external pull-up)
- = 1 for either an open drain output or input

After P5_MODE is set, the 8XC196MC automatically determines SF5_DIR, SF_SET and SF_$\overline{\text{RESET}}$.

For example, suppose you want to use the special function signals $\overline{\text{BHE}}$ (P5.5) and BUSWIDTH (P5.7). $\overline{\text{BHE}}$ is an active-low output. If you are not using an external pull-up for it, you should write a 0 to bit 5 of P5_DIR and a 1 to bit 5 of P5_REG. BUSWIDTH is an input. Thus, you should write 1's to bit 7 of P5_DIR and P5_REG.

The proper settings of P5_DIR and P5_REG for a desired configuration and value of P5_PIN are given in Figure 10.5.

10.2.4 Port 5 Special Function Signals

At reset the Port 5 pins are driven to a weak 1 to avoid undefined states on pins (such as $\overline{\text{WR}}$) and to avoid excessive current due to floating inputs. After reset the 8XC196MC configures itself to match the external system. The following paragraphs describe the states of the port 5 pins after reset and until the user writes to the mode register P5_MODE. You should write to P5_MODE even if all port 5 pins are used as I/O port pins, because writing to this register will disable the weak pull-ups, thus reducing leakage and minimizing input switching currents.

Writing a 1 to P5_MODE.x configures the pin as a special function. Writing a 0 configures the pin for I/O.

NOTE

The Port 5 special functions are briefly described in Appendix A. ALE, $\overline{WR}$, $\overline{RD}$, $\overline{BHE}$, READY and BUSWIDTH are associated with external memory interfacing and are described in Section 15.

ALE, $\overline{RD}$. If $\overline{EA}$ = 1 on reset (internal access), ALE and $\overline{RD}$ stay at a weak 1 until a write to P5_MODE register is performed. If $\overline{EA}$ = 0 on reset (external access), ALE and $\overline{RD}$ are activated as system control pins and become true complementary outputs. ALE can be put into its secondary system function $\overline{ADV}$, depending on the chip configuration byte CCB0 (Section 15.2).

$\overline{BHE}$. $\overline{BHE}$ stays at a weak 1 until the chip configuration byte fetch is completed. Once the CCB0 fetch is done, the state of $\overline{BHE}$ depends on the value of the BW0 bit:

- If BW0 = 0, $\overline{BHE}$ continues at a weak 1 until a write to P5_MODE is performed
- If BW0 = 1, $\overline{BHE}$ becomes a true complementary output

$\overline{WR}$, INST. These pins stay at a weak 1 until a write to P5_MODE is performed.

READY. This pin stays at a weak 1 until the chip configuration byte fetch is completed. Thereafter, the state depends on bits IRC0-2 of the chip configuration bytes. These bits and the READY signal control the wait states (Section 15.3.1). On reset, if IRC0-2 = 111B, P5.6 is configured as the READY special function. This prevents infinite wait states from being inserted upon the first access to external memory. For all other combinations of IRC, the pin will be configured as standard I/O upon reset.

NOTE

If IRC0–2 = 111B and P5_MODE.6 is cleared (configured for standard I/O) an external memory access will cause the processor to lock-up.

BUSWIDTH. BUSWIDTH is weakly forced to a 1 until a write to the P5_MODE register is performed.

Interrupts 11

CHAPTER 11
INTERRUPTS

An interrupt is a change in the program flow. Instead of executing the next instruction, the CPU branches to an interrupt service routine. The branch occurs in response to a request from an on-chip peripheral, an external signal or an instruction. In the simplest case, the 8XC196MC receives the request, performs the service and returns to the task that was interrupted. This section discusses interrupts that are serviced by software interrupt service routines. At the user's choice, most interrupts can instead be serviced by hardware via the Peripheral Transaction Server (PTS) described in Section 12.

Hardware interrupts (those initiated by an on-chip peripheral or an external signal) follow the procedure illustrated in Figure 11.1. The transition detector recognizes an interrupt request (a rising edge on an interrupt signal), and sets a bit associated with that interrupt source in the interrupt pending register. Other bits in this register may already be set, indicating prior interrupts received from other sources. For a particular interrupt source to be considered for service, two further conditions must be met. First, the user must have set the global interrupt enable bit to permit servicing **any** interrupt. Also, the user must have set the bit in the interrupt mask register that corresponds to the particular source. Thus, the AND gate in the figure signifies that an interrupt is considered for service only if:

- the corresponding bit is set in the Interrupt Pending Register (INT_PEND or INT_PEND1), AND
- the corresponding bit is set in the Interrupt Mask Register (INT_MASK or INT_MASK1), AND
- the Global Interrupt Enable Bit is set in the Program Status Word (PSW.9)

All interrupts meeting these conditions are passed to the priority encoder, which selects the interrupt with the highest priority. The interrupt priorities are set by the hardware, but they can be modified with software as shown in Section 11.3. Each interrupt has an assigned address in the interrupt vector table. The address (interrupt vector) is the starting address of the interrupt service routine. When the interrupt is serviced, control is transferred to the instruction at that address. Subsequent sections describe the details of this process and how the user can control it with software.

The non-maskable interrupt (NMI, Section 11.2) is handled differently. This external hardware interrupt has the highest priority, and it **cannot** be blocked by the interrupt mask register or the global interrupt enable bit (PSW.9).

Table 11.1 lists the interrupts in their priority order (15 is highest) along with references to sections that describe the associated sources. Each of the sources represents a single interrupt. Thus, the 8XC196MC has a total of 16 interrupts and 15 interrupt vectors.

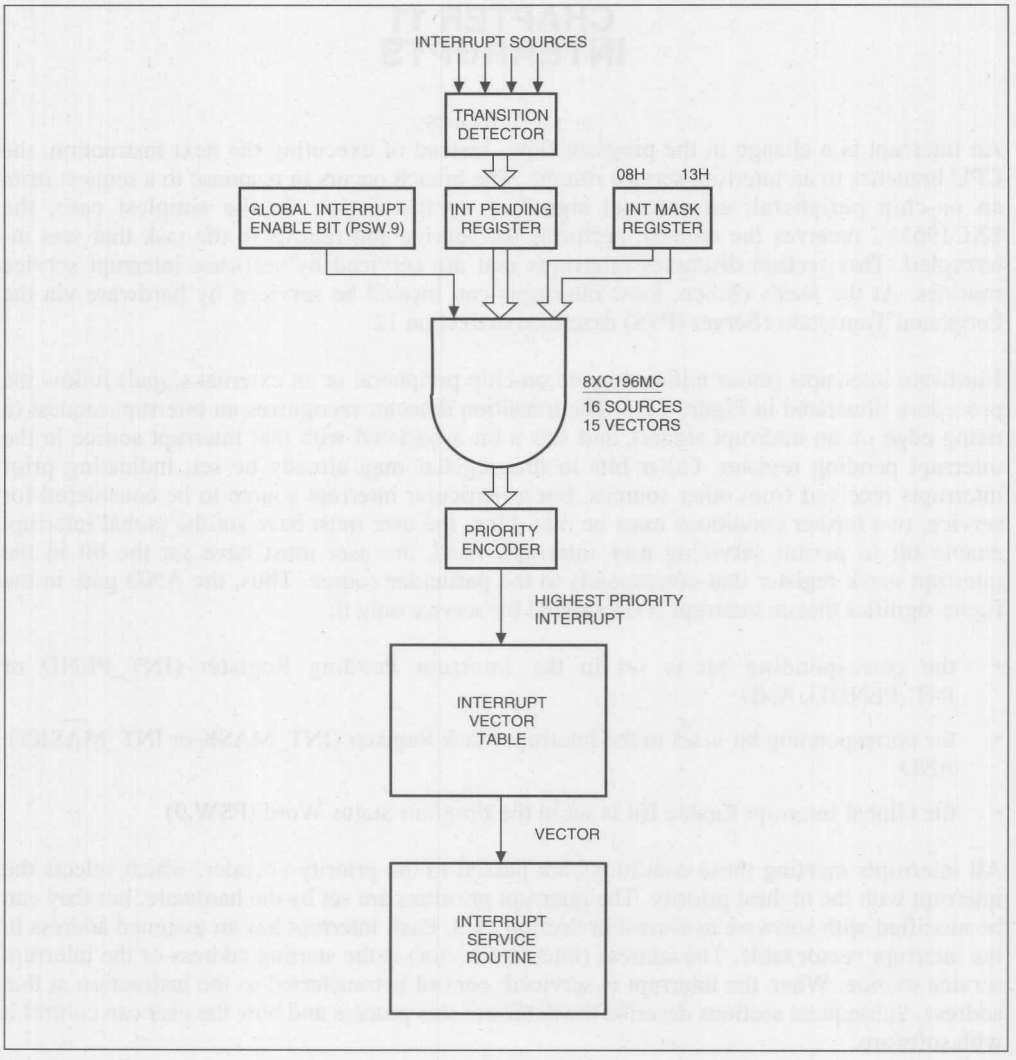


Figure 11.1. 8XC196MC Interrupt Structure Block Diagram

Table 11.1. Summary of Interrupts and Priorities

Number	Source	Symbol	Vector Locn.	Priority
INT15	Non-Maskable Interrupt	NMI	203EH	15
PTS14-PTS0	PTS	*	*	*
INT14	EXTINT Pin	EXTINT	203CH	14
INT13	PI	PI	203AH	13
INT12	Reserved	—	2038H	12
INT11	Reserved	—	2036H	11
INT10	Reserved	—	2034H	10
INT09	COMPARE3	CM3	2032H	9
INT08	CAPCOMP3	CP3	2030H	8
N/A	Unimplemented Opcode	N/A	2012H	N/A
N/A	TRAP Instruction	N/A	2010H	N/A
INT07	COMPARE2	CM2	2003H	7
INT06	CAPCOMP2	CP2	200CH	6
INT05	COMPARE1	CM1	200AH	5
INT04	CAPCOMP1	CP1	2008H	4
INT03	COMPARE0	CM0	2006H	3
INT02	CAPCOMP0	CP0	2004H	2
INT01	A/D Complete	A/D DONE	2002H	1
INT00	T1/T2 Overflow/Underflow	TOVF	2000H	0

*All PTS requests have higher priority than normal interrupt requests.

Section 11.1 describes interrupt control. Three special interrupts are discussed in Section 11.2. Section 11.3 covers the interrupt priorities and how they can be modified by user software. Interrupt timing is discussed in Section 11.4. Section 12 describes the peripheral transaction server (PTS), which is also intimately involved with interrupts.

Interrupt SFRs. The Interrupt SFRs are listed with their addresses:

SFR	Address
INT_MASK	08H
INT_PEND	09H
INT_PEND1	12H
INT_MASK1	13H
PI_PEND	1FBEH
PI_MASK	1FBCH

Interrupt External Signals. There are two external interrupt signals:

NMI	Non-Maskable interrupt
EXTINT	External Interrupt

External sources must have a pulse width of at least one state time to guarantee that they will be seen by the transition detector.

11.1 INTERRUPT CONTROL

11.1.1 Interrupt Pending Register

The interrupt pending registers INT_PEND and INT_PEND1 contain the bits that are set when a hardware interrupt is detected. Figure 11.2 shows these registers, with the bit codes corresponding to the 16 hardware sources given in Table 11.1. When the interrupt vector is taken, the pending bit is cleared. INT_PEND and INT_PEND1 can be read to determine which of the interrupts are pending at any given time or modified, either to clear pending interrupts or to generate interrupts under software control.

INT_PEND1/INT_MASK1 (12H/13H)

7	6	5	4	3	2	1	0
NMI	EXTINT	PI	reserved	reserved	reserved	COMPARE3	CAPCOMP3

Figure 11.2a. Interrupt Pending and Mask Registers

INT_PEND/INT_MASK (09H/08H)

7	6	5	4	3	2	1	0
COMPARE2	CAPCOMP2	COMPARE1	CAPCOMP1	COMPARE0	CAPCOMP0	A/D DONE	TIMER OVF

Figure 11.2b. Interrupt Pending and Mask Registers

PI_PEND/PI_MASK (1FBEH/1FBCH)

7	6	5	4	3	2	1	0
reserved	reserved	reserved	WG	reserved	TF2	reserved	TF1

Figure 11.2c. Peripheral Interrupt Pending and Mask Registers

Care should be taken in writing code that modifies these registers. For example, an instruction sequence to clear a pending bit could result in an interrupt being acknowledged after the sequence begins but before the bit is actually cleared. In this case a 5 state time *partial* interrupt cycle occurs. That is, the interrupt process begins but stops short of jumping to the interrupt service routine. This time delay can be avoided by making the code inseparable in the sense that an interrupt will not be acknowledged while the code is executing. The easiest way to do this is to use the logical instructions in the two- or three-operand format, for example:

```
ANDB   INT_PEND, #01111111B      ; This clears the COMPARE2 interrupt
ORB    INT_PEND, #10000000B      ; This sets the COMPARE2 interrupt
```

The 8XC196MC holds off acknowledging interrupts during these ''read-modify-write'' instructions.

11.1.2 Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask registers INT_MASK and INT_MASK1. The format of these registers is the same as for the interrupt pending registers shown in Figure 11.2.

A 1 in any bit position enables the corresponding interrupt source; a 0 disables the source. An exception is the non-maskable interrupt (NMI). **The NMI bit in INT_MASK1 is non-functional; the non-maskable interrupt is enabled for NMI equal to 0 or 1.** For compatibility with future products, the NMI bit should always be 0.

After RESET, the interrupt mask registers are cleared (interrupts disabled). The hardware saves any interrupts that occur by setting bits in the pending registers, even if the interrupt mask registers are cleared.

The PUSHF and POPF instructions save and restore the INT_MASK register as well as the PSW. INT_MASK1 is the byte above the window select register (WSR) and is saved and restored along with the WSR by the PUSHA and POPA instructions. PUSHF and PUSHA clear the PSW and the interrupt mask, thus disabling the interrupts.

11.1.3 Global Interrupt Enable

The processing of all interrupts except the NMI, TRAP, and unimplemented opcode interrupts can be disabled by clearing the I bit in the PSW (bit 9). Setting the I bit enables interrupts that have mask register bits set. The I bit is controlled by the EI (enable interrupts) instruction and DI (disable interrupts) instruction. Note that the I bit only controls the servicing of interrupts. Interrupt requests that occur when the I bit is cleared are held in the interrupt pending registers. If the corresponding mask bit is set, each interrupt is serviced according to its priority.

11.2 SPECIAL INTERRUPTS

Three special interrupts are available on the 8XC196MC: NMI, TRAP and Unimplemented Opcode. Although available to the user, these interrupts may be used by Intel evaluation or emulation tools. For this reason, these interrupts should not be used unnecessarily.

NMI

The external non-maskable interrupt (NMI), has the highest priority. For design symmetry, a mask bit for NMI exists in INT_MASK1, but the bit does **not** mask out NMI interrupts and will **not** stop an NMI from occurring. For future compatibility, the NMI mask bit should be set to zero.

NMI on the 8096BH vectored directly to location 0000H, whereas all interrupts to the 8XC196MC vector indirectly via the vector locations shown in Table 11.1. For software to be compatible with 8096BH software that uses the NMI, location 203EH should be loaded with 0000H. The NMI interrupt vector and interrupt vector location may be used by some development tools. The NMI pin should be connected to V_{SS} when not used.

TRAP

The TRAP instruction provides a single instruction interrupt useful in designing software debuggers or software interrupts. TRAP prevents the acknowledgement of interrupts until after execution of the next instruction.

Unimplemented Opcode

Execution of opcodes that are not implemented cause an interrupt. The programmer must initialize the interrupt vector table with the starting address of the appropriate interrupt service routine. The error routine should contain recovery code that will not further corrupt an already erroneous situation.

11.3 INTERRUPT PRIORITIES

The priority encoder looks at all of the interrupts that are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Table 11.1. (15 is the highest; 0 is the lowest.) The interrupt then forces a call to the address in the indicated vector location. This address is the starting location of the interrupt service routine (ISR).

This priority selection controls the order in which pending interrupts are passed to the software via interrupt calls. The software can then implement its own priority structure by controlling the mask registers (INT_MASK and INT_MASK1). To see how this is done, consider the case of a Timer 1 interrupt service routine which must run at a priority level that is higher than the levels of any other interrupt sources except for the EXTINT interrupt. The EXTINT interrupt is

to have the highest priority. The ''preamble'' and exit code for this routine would look like this:

```
TIMER1_ISR:

    PUSHA                          ;Save the PSW
    LDB     INT MASK1, #01000000B
    EI                             ;Enable Interrupts again and allow only
                                   ;EXTINT interrupts.
                                   ;
                                   ;Service the TIMER1 Interrupt
                                   ;
    POPA                           ;Restore the PSW and old INT_MASK1
    RET
```

Note that location 2000H in the interrupt vector table would be loaded with the value of the label TIMER1_ISR, and that the interrupt must be enabled for this routine to execute.

An interesting chain of instruction side-effects makes this (or any other) interrupt service routine execute properly:

1. After the hardware decides to process an interrupt, it generates and executes a special interrupt call. This pushes the program counter onto the stack and then loads it with the contents of the vector corresponding to the highest priority interrupt that is pending and unmasked. The hardware will not allow another interrupt to be serviced immediately following the interrupt call instruction. This guarantees that once the interrupt call starts, the first instruction of the interrupt service routine is executed.
2. The PUSHA instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears it. In addition to the arithmetic flags, the PSW contains INT_MASK and the global interrupt enable bit (I). PUSHA also stacks INT_MASK1 and WSR. INT_MASK1 is cleared when pushed. This allows the interrupt service routine to manipulate the WSR without affecting the interrupted routine. The hardware will not allow an interrupt following the PUSHA instruction. By the time the LDB instruction starts, all of the interrupt enable flags have been cleared. Now there is guaranteed execution of the LDB INT_MASK1 instruction.
3. The LDB INT_MASK1 instruction enables those interrupts that the programmer chooses to allow to interrupt the TIMER1_ISR. In this example only the EXTINT is allowed to interrupt the TIMER1 service routine. At this point any interrupt or interrupt combination could be enabled, including the timer interrupts. The loading of the INT_MASK1 register allows the software to establish its own priorities for interrupt servicing independent of those enforced by the hardware.
4. The EI instruction re-enables interrupt processing by setting the Global Interrupt enable bit (PSW.9).
5. The actual interrupt service routine executes within the priority structure established by the software.

6. At the end of the service routine the POPA instruction restores the original PSW, the interrupt mask registers, and the WSR. As the hardware prevents processing of interrupts following a POPA instruction, execution of the last instruction (RET) is guaranteed before further interrupts can occur. The RET instruction must be protected in this fashion because it is quite likely that the POPA instruction will re-enable an interrupt that is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts occur at a high frequency. The POPA instruction also pops the INT_MASK and INT_MASK1 registers. So, any changes made to this register during a routine which ends with a POPA are lost.

Notice that the "preamble" and exit code for this routine does not save or restore register RAM. The interrupt service routine is assumed to allocate its own private set of registers from the Lower Register File. The availability of 232 bytes of RAM in the Lower Register File makes this quite practical. In addition, the 256 bytes of the Upper Register File are available via windowing (Section 4.3).

11.4 INTERRUPT TIMING

The 8XC196MC can be interrupted from two external sources (other than the EPA). These sources (EXTINT and NMI) are sampled during the Phase 1 half of the state time (CLKOUT low) and are latched internally. Holding a level on an external interrupt for at least one state time ensures recognition of that interrupt.

Although sampled during Phase 1, hardware interrupts on the 8XC196MC are edge triggered as opposed to level triggered. Edge triggered interrupts generate only one interrupt if the input is held high. On the other hand, level triggered interrupts would generate multiple interrupts when held high/low.

There is a delay between an interrupt's triggering and its acknowledgement. An interrupt is not acknowledged until the currently executing instruction is finished. Further, if the interrupt signal does not occur at least 4 state times before the end of the current instruction, the interrupt may not be acknowledged until after the **next** instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed. Thus, the maximum delay between interrupt generation and acknowledge–ment is 4 state times plus the execution time of the next instruction.

There are six instructions that always inhibit interrupt acknowledgement until after the following instruction has been executed: EI, DI, PUSHF, POPF, PUSHA and POPA. Also, interrupts cannot occur immediately after execution of:

- Unimplemented Opcodes
- TRAP, the software trap instruction
- The signed prefix opcode for the signed multiply and divide instructions

In these situations, the generation-to-acknowledgement delay is increased.

When an interrupt is acknowledged the interrupt pending bit is cleared and a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the current instruction, except as noted above. The procedure of getting the vector and forcing the call requires 16 state times. If the stack is in external RAM, an additional 2 state times are required. This assumes a zero wait state bus.

Latency is the time from when the interrupt is generated (not acknowledged) until the 8XC196MC begins executing interrupt code. The maximum latency occurs when the interrupt occurs too late for acknowledgement following the current instruction. The worst case is calculated assuming that the current instruction is not one of those listed above that inhibit acknowledgement. The sum consists of three terms:

1. 4 state times for the current instruction to finish.
2. The state times for the next instruction; the longest instruction (NORML) takes 39 state times.
3. The response time (16 state times for an internal stack or 18 for an external stack).

Thus the maximum delay is 4 + 39 + 18 = 61 state times. This does not include the 10 state times required for PUSHF or PUSHA if it is the first instruction in the interrupt service routine. Figure 11.3 illustrates an example of this worst case scenario.

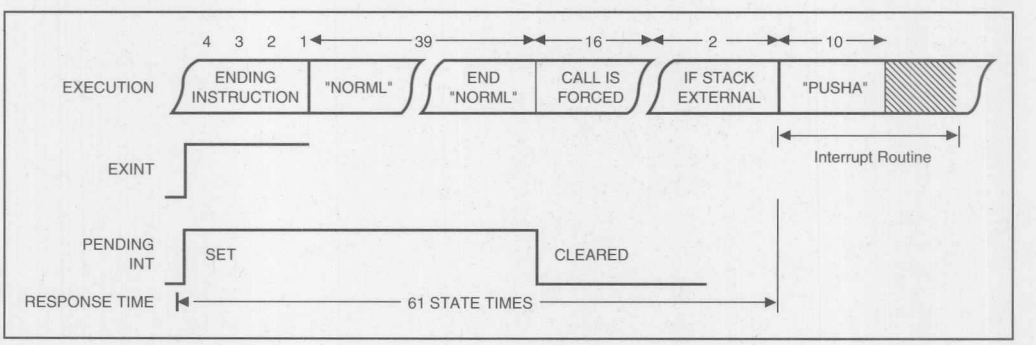


Figure 11.3. Interrupt Response Time

Interrupt latency can be reduced by carefully selecting instructions in areas of code where interrupts are expected. Using EI followed immediately by a long instruction (e.g., MUL, DIV, NORML) increases the maximum latency by 4 states because an interrupt cannot occur between EI and the instruction following EI. The DI, PUSHF, PUSHA, POPF, POPA and TRAP instructions also have the same effect. Typically, these instructions would only affect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

Peripheral Transaction Server (PTS) 12

Peripheral Transaction Server (PTS) 12

CHAPTER 12
PERIPHERAL TRANSACTION SERVER (PTS)

The Peripheral Transaction Server (PTS) provides a microcoded hardware interrupt handler. It can be used in place of a normal interrupt service routine. Interrupts can be serviced in the time required to execute a single instruction. The PTS execution is interleaved with the normal instruction flow through a cycle stealing operation. Thus, an advantage of the PTS over normal interrupt operation is its ability to service interrupts with less overhead. It operates without modification of the stack or the PSW and allows normal instruction flow to continue.

A further advantage of the PTS is its ability to operate in any of several modes designed for specialized operations. These include:

- Single byte/word transfers in memory
- Block byte/word transfers in memory
- Management of multiple A/D conversions on different channels
- Synchronous and asychronous serial input output

Any interrupt source, except NMI, TRAP and Unimplemented Opcode, can be used with the PTS. Setting a bit in the PTS select register (PTSSEL) selects the corresponding interrupt for PTS service.

The PTS is closely associated with the normal interrupt system. Section 11 on Interrupts should be read before trying to understand the PTS.

An interrupt is handled by the PTS if the global PTS enable bit is set and if the PTS is selected for the interrupt source. The flow diagram of Figure 12.1 summarizes the operation of the PTS and its coordination with the normal interrupt system. An interrupt signal rising edge is recognized by the transition detector, and the corresponding bit in the interrupt pending register is set. For all interrupts except NMI, Unimplemented Opcode and Trap the interrupt mask bits are checked. Any interrupt whose mask bit is 0 is blocked. If the mask bit is 1, the corresponding bit in PTSSEL is examined. If the PTS select bit = 1, the interrupt is serviced by the PTS; if the PTS select bit = 0, the interrupt is treated normally. All PTS requests are sent on to the priority encoder unless the global PTS enable bit = 0. All normal interrupt requests are sent to the priority encoder unless the global interrupt enable bit = 0.

The priority encoder determines the interrupt with the highest priority. All PTS requests have higher priority than normal interrupt requests. If PTS requests are pending, the one with highest priority is passed to the PTS vector table. The corresponding vector transfers control to the PTS control block. The control block determines the PTS mode and the details of handling the transaction. If no PTS request is pending, then the normal interrupt with the highest priority is serviced.

Subsequent sections describe PTS control and timing and the PTS operation modes.

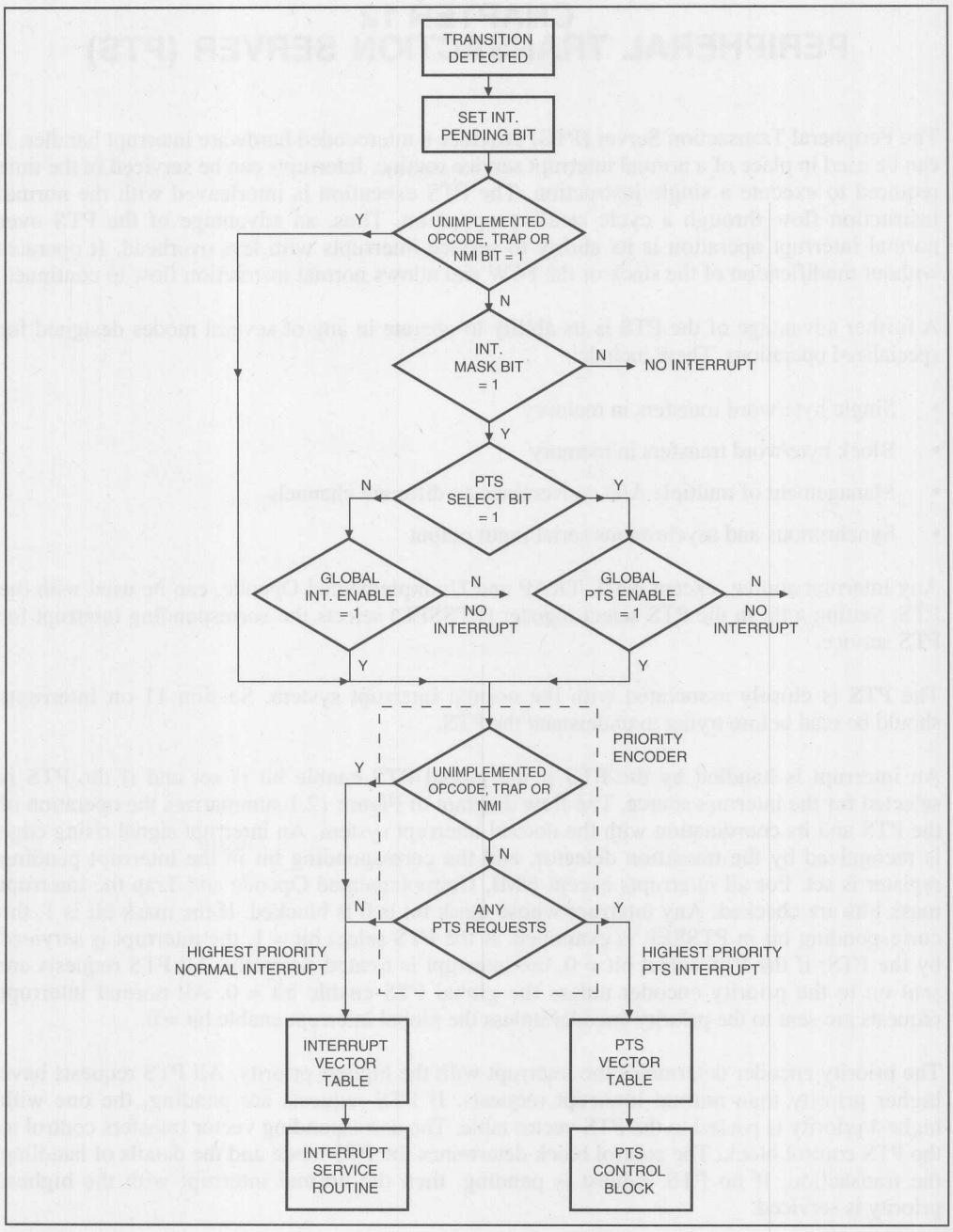


Figure 12.1. Flow Diagram for PTS and Normal Interrupts

PTS SFRs. The PTS has two 16-bit SFRs:

SFR	Address
PTSSRV	06H
PTSSEL	04H

PTS External Signals. The external signals associated with the PTS are EXTINT and each capture/compare pin.

12.1 PTS CONTROL

The global PTS enable bit PSE (PSW.10) determines whether PTS interrupts are enabled (PSE = 1) or disabled (PSE = 0). This bit is set/cleared by the enable/disable instructions (EPTS/DPTS). Setting a bit in PTSSEL (Figure 12.2) designates the corresponding interrupt as a PTS interrupt. The format is the same as for the interrupt pending and interrupt mask registers (Figure 11.2), except for bit 15, which is reserved. Table 12.1 shows the table of PTS priorities and vector locations. The priority order is the same as for normal interrupts, except that NMI is omitted. The PTS vector points to the first location of the PTS control block (PTSCB) for that PTS interrupt. The PTSCB (described in Section 12.3) specifies the microcode to be executed. The PTS then executes a PTS cycle. The PTSCB determines the type of PTS cycle executed.

PTSSEL (04H) or PTSSRV (06H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rsv	EXT INT	WG	rsv	rsv	rsv	COMP3	CAP/ COMP3	COMP2	CAP/ COMP2	COMP1	CAP/ COMP1	COMP0	CAP/ COMP0	A/D DONE	Timer over-flow

Refer to Section 11 (Interrupts) for bit descriptions.

Figure 12.2. The PTS_SEL and PTS_SRV Registers

When an interrupt is serviced by the PTS, the priority encoder selects the appropriate PTS vector, and the PTSCB is fetched. The PTSCB determines what type of PTS cycle is executed. When the PTS has finished servicing the interrupt, a series of events occurs.

1. The PTS count is decremented.
2. If the count is not 0, nothing happens. (Skip steps 3 and 4.)
3. If the count is 0, the associated bit in the PTSSEL register is cleared to inhibit any further PTS cycles until the bit is set again.
4. The associated bit in the PTS service register PTSSRV is set to indicate an end of the PTS interrupt.

PTSSRV is a register with the same format as PTSSEL (Figure 12.2). PTSSRV holds requests for an ''end-of-PTS'' just as the interrupt pending registers hold requests for interrupts.

End-of-PTS interrupts are used to indicate that the PTS must be serviced. PTSSRV is used instead of the pending registers to differentiate end-of-PTS interrupts from normal interrupts. This allows an interrupt request to be buffered in the pending register until the PTS is serviced. Servicing the PTS includes reinitializing the PTSCB and setting the appropriate PTSSEL bit. After the PTS is serviced, it will take care of the pending interrupt request.

Table 12.1. PTS Vector Table

PRIORITY	NAME	SOURCE	PTS VECTOR
HIGHEST	PTS14	EXTINT	205CH
•	PTS13	WG	205AH
•	PTS12	RESERVED	2058H
•	PTS11	RESERVED	2056H
•	PTS10	RESERVED	2054H
•	PTS9	COMPARE3	2052H
•	PTS8	CAPCOMP3	2050H
•	PTS7	COMPARE2	204EH
•	PTS6	CAPCOMP2	204CH
•	PTS5	COMPARE1	204AH
•	PTS4	CAPCOMP1	2048H
•	PTS3	COMPARE0	2046H
•	PTS2	CAPCOMP0	2044H
•	PTS1	A/D Done	2042H
LOWEST	PTS0	TOVF	2040H

The end-of-PTS interrupt is treated as a normal interrupt. It vectors through the associated location in the normal interrupt vector table (Table 11.1). For example, if the A/D interrupt is selected by the PTS, an A/D interrupt is directed to its PTSCB by the PTS vector at 2042H; its end-of-PTS interrupt is at 2002H. Thus, the user would write an end-of-PTS interrupt routine for A/D DONE and store a vector pointing to it at location 2002H.

An end-of-PTS interrupt has higher priority than any normal interrupt (with the exception of NMI). Within the group of end-of-PTS interrupts, the priorities are the same as for normal interrupts. For example, suppose the following interrupts are pending and that no others become pending while these are serviced:

Symbol	Priority	Type
EXTINT	14	Normal
A/D DONE	12	End-of-PTS
COMPARE0	10	End-of-PTS

The order of servicing the interrupts is A/D DONE, COMPARE0 and lastly EXTINT.

When the end-of-PTS interrupt is called, the bit in the PTSSRV is automatically cleared; however, the PTSSEL bit must be set manually to re-enable the PTS channel.

12.2 PTS TIMING

Because the preludes to a PTS request and a normal interrupt request are so similar, the latency (time delay) in beginning a PTS operation is calculated in the same way as for a normal interrupt (see Section 11.4). The maximum delay between an interrupt signal rising edge and its acknowledgement is 43 state times as shown in Figure 12.3. This latency does not include the added delay incurred if:

1. the PTS is disabled because the PSE bit = 0, or
2. any higher priority PTS request is executing

In the second case the delay is longer if a block PTS cycle is executing.

Once the PTS request is acknowledged, the response time is equal to the PTS execution time.

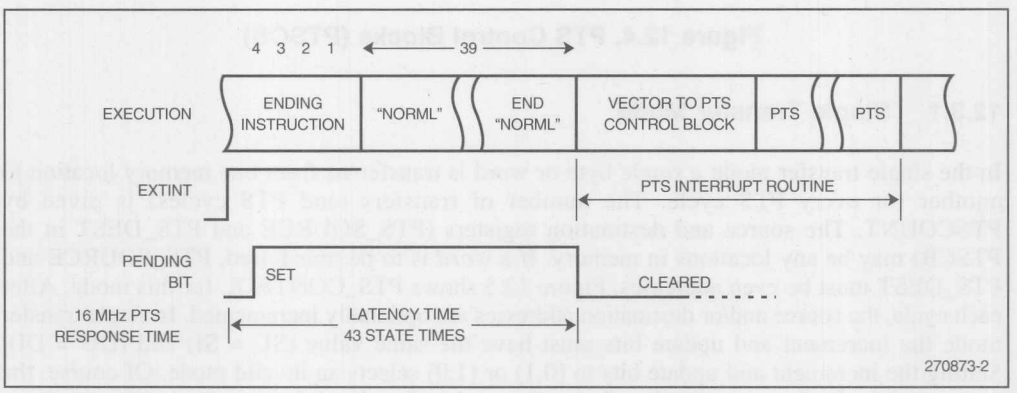


Figure 12.3. PTS Latency

12.3 PTS MODES

The PTS mode is selected by the PTS control block (PTSCB). The PTSCB formats for the five modes are shown in Figure 12.4. Although the formats differ, they share many features. The PTSCB for a particular interrupt source must be set up by user software before that PTS interrupt is enabled. It must be in register RAM and the first (lowest) byte must be at an address evenly divisible by 8 (quad word boundary). This first address (PTSVECTOR) is stored in the PTS vector table (Table 12.1).

The first byte in the PTSCB, PTSCOUNT, defines the number of PTS cycles to be executed without CPU intervention. Since PTSCOUNT is an 8-bit value, the maximum number of

cycles is 256. At the end of each PTS cycle, PTSCOUNT is decremented. When PTSCOUNT is decremented to zero, the PTSSRV bit is set to request an end-of-PTS as described in Section 12.1. Bytes in the PTSCB labeled UNUSED are available as extra RAM; their contents are neither read nor written by the PTS.

The second byte of the PTSCB, PTS_CONTROL, determines the PTS mode and parameters for that mode. The formats for PTS_CONTROL are given in following sections on the different modes.

UNUSED	UNUSED	UNUSED	PTSVEC1 (HI)	UNUSED
UNUSED	PTS_BURST	UNUSED	PTSVEC1 (LO)	SAMPTIME
PTS_DEST (HI)	PTS_DEST (HI)	PTS_PTR2 (HI)	BAUD (HI)	DATA (HI)
PTS_DEST (LO)	PTS_DEST (LO)	PTS_PTR2 (LO)	BAUD (LO)	DATA (LO)
PTS_SOURCE (HI)	PTS_SOURCE (HI)	PTS_PTR1 (HI)	EPAREG (HI)	PTSCON1
PST_SOURCE (LO)	PTS_SOURCE (LO)	PTS_PTR1 (LO)	EPAREG (LO)	PORTMSK
PTS_CONTROL	PTS_CONTROL	PTS_CONTROL	PTSCON	PORTREG (HI)
PTS VECTOR	PTSCOUNT	PTSCOUNT	PTSCOUNT	PORTREG (LO)
Single Transfer	Block Transfer	A/D	SIO	SIO

Figure 12.4. PTS Control Blocks (PTSCB)

12.3.1 Single Transfer Mode

In the single transfer mode a single byte or word is transferred from one memory location to another for every PTS cycle. The number of transfers (and PTS cycles) is given by PTSCOUNT. The source and destination registers (PTS_SOURCE and PTS_DEST in the PTSCB) may be any locations in memory. If a *word* is to be transferred, PTS_SOURCE and PTS_DEST must be even addresses. Figure 12.5 shows PTS_CONTROL for this mode. After each cycle, the source and/or destination addresses are optionally incremented. In single transfer mode the increment and update bits must have the same value (SU = SI) and (DU = DI). Setting the increment and update bits to (0,1) or (1,0) selects an invalid mode. Of course, the source and destination can be optionally incremented (and updated) independently of each other.

For example, if you want to copy a word at location 20H to locations 6000H, 6002H, ..., 6008H, you would use:

PTSCOUNT	= 5	SU	= 0
PTS_SOURCE	= 0020H	DU	= 1
PTS_DEST	= 6000H	SI	= 0
B/$\overline{W}$	= 0	DI	= 1

Single transfer mode is typically used for loading the waveform generator control register (WG_CON), PWM register or a compare register of the EPA. It can also be connected to the EPA in capture mode to move the captured time from the SFR location to internal RAM for further processing.

PTS_CONTROL

7	6	5	4	3	2	1	0
M2	M1	M0	B/$\overline{W}$	SU	DU	SI	DI

M2, M1, M0	Mode	Function
	100	Single Transfer
B/$\overline{W}$	Byte (1) / Word (0) Transfer	
SU	1 = Update PTS_SOURCE after each PTS cycle	
DU	1 = Update PTS_DEST after each PTS cycle	
SI	1 = Autoincrement PTS_SOURCE after each PTS cycle	
DI	1 = Autoincrement PTS_DEST after each PTS cycle	

NOTE: SI = SU and DI = DU for Single Transfer

Figure 12.5. PTS Control for Single Transfer

12.3.2 Block Transfer Mode

In block transfer mode a block of data can be moved during each PTS cycle. The number of bytes/words in the block, which may range from 1 to 32, is specified by PTS_BURST in the PTSCB. Figure 12.6 shows PTS_CONTROL for block transfer. In this mode it is important to differentiate between a *transfer* and a *PTS cycle*. A *transfer* is the movement of a single byte/word from source to destination. A *PTS cycle* consists of PTS_BURST consecutive transfers. If the increment bit is set, the source and/or destination addresses are incremented after each transfer. If the update bit is set, the incremented address is saved after each cycle. Setting both increment and update means that the source and/or destination address will be continuously incremented until all cycles are complete. The increment and update may be selected independently (unlike in single transfer mode).

For example, assume you want to copy the bytes in the memory block 20H-24H to the three blocks 6000H-6004H, 6005H-6009H and 600AH-600EH. This is a *byte* transfer. It requires a burst of 5 transfers per cycle and 3 cycles. The source and destination are incremented after each transfer. Only the destination is updated after each cycle; the first byte of each cycle is read from 20H. The parameters are:

PTSCOUNT	= 3	B/$\overline{W}$	= 1
PTS_BURST	= 5	SU	= 0
PTS_SOURCE	= 0020H	DU	= 1
PTS_DEST	= 6000H	SI	= 1
		DI	= 1

As a single PTS cycle is uninterruptable, the user must be aware of the potential for long interrupt latency in block transfer mode. In a worst case – a block transfer mode of 32 words from external memory to external memory in 8-bit addressing bus mode with no wait states inserted – a maximum latency of 500 states can be expected.

PTS_CONTROL

7	6	5	4	3	2	1	0
M2	M1	M0	B/$\overline{W}$	SU	DU	SI	DI

M2, M1, M0	Mode	Function
	100	Single Transfer
B/$\overline{W}$	Byte (1) / Word (0) Transfer	
SU	1 = Update PTS_SOURCE after each PTS cycle	
DU	1 = Update PTS_DEST after each PTS cycle	
SI	1 = Autoincrement PTS_SOURCE after each PTS cycle	
DI	1 = Autoincrement PTS_DEST after each PTS cycle	

Figure 12.6. PTS Control for Block Transfer Mode

12.3.3 A/D Scan Mode

The A/D scan mode facilitates performing A/D conversions on single or multiple channels and storing the results. The user first sets up a table of the A/D commands, with locations reserved for the results. The PTS then automatically starts the conversions and stores the results after software initiates the first conversion.

In the PTSCB for the A/D scan mode (Figure 12.4), PTS_COUNT specifies the number of PTS cycles, which is normally the number of A/D conversions desired without an interrupt. The PTS_CONTROL format for the scan mode is shown in Figure 12.7. PTS_PTR1 points to the table of conversion commands and results. PTS_PTR2 points to the A/D command/control register.

PTS_CONTROL

7	6	5	4	3	2	1	0
M2	M1	M0	B/$\overline{W}$	SU	DU	SI	DI
1	1	0	0	SU	0	1	1

SU = 0 for no update of PTS_PTR1 after the PTS cycle (saves old value)
= 1 for update of PTS_PTR1 after the PTS cycle (saves new value = old value + 4)
All other bits are fixed.

Figure 12.7. PTS_CONTROL for A/D Scan Mode

The sequence of actions for a single PTS cycle in the A/D scan mode is:

Description	Action in A/D Scan Mode
1. The word specified by pointer PTS_PTR1 is read and saved in a temporary location in the RALU.PTS_PTR1 is incremented by 2.	PTS_PTRI points to a command in a table containing alternating commands and data words. Command_x is read from the table and stored in a temporary register. PTS_PTR1 is incremented by 2 and points to where AD_RESULT from the previous conversion will be stored.
2. The word specified by pointer PTS_PTR2 is read and stored in the location specified by PTS_PTR1. PTS_PTR1 is incremented by 2.	PTS_PTR2 points to the AD_RESULT register. AD_RESULT is read and stored in the command/data table. PTS_PTR1 is incremented by 2 and points to (Command_x + 1) in the command/data table.
3. The lower byte of the word saved in step 1 is stored in the location specified by PTS_PTR2.	Command_x stored in step 1 is written to AD_RESULT. When the PTS writes to AD_RESULT, the value actually goes into AD_COMMAND (only if the write is performed by the PTS). This starts the next A/D conversion cycle.
4. If SU = 0, the old contents of PTS_PTR1 (the value before step 1 of the cycle) are saved. If SU = 1, the new contents of PTS-PTR! (old value + 4) are saved.	If SU = 0, PTS_PTR! is restored to point to Command_x. The next cycle will use the same command and overwrite previous data. If SU = 1, PTS_PTR1 points to the next command after Command_x.
5. PTS_COUNT is decremented. The cycle is repeated until PTS_COUNT = 0.	Steps 1-5 are repeated until PTS_COUNT = 0.

As illustrated in the examples below, effective use of this sequence for handling A/D commands and results rests on the fact that *in the A/D Scan Mode a write to the AD_RESULT register actually performs a write to the AD_COMMAND register.* This is a property of the AD_RESULT and AD_COMMAND registers when written by the PTS and not a property of the PTS. This redirection does not work with other registers or when writing to AD_RESULT with software.

The following example shows how to set up a series of A/D conversions and reads, beginning with channel 7 and ending with channel 0. First, set up a table of A/D conversion commands as shown in Table 12.2, except that the entries marked ''AD_RESULT for ACHx'' are absent; they will be filled in upon execution of the PTS scan mode. Each entry in the table is a word (2 bytes). Next, initialize PTS COUNT, PTS_CONTROL and the pointers:

Address	Contents
PTS COUNT	08H
PTS_CONTROL	0CBH (update SU)
PTS_PTR1	3000H
PTS–PTR2	1FAAH (AD_RESULT SFR)

User software starts a conversion on channel 7. Upon completion of the conversion, the A/D interrupt initiates the 5-step sequence above. Step 1 stores the AD_COMMAND for channel 6 in the ALU and increments PTS_PTR1 to 3002H. Step 2 stores the result of the channel 7 conversion in location 3002H and increments PTS_PTR1 to 3004H. Step 3 loads the AD_COMMAND for channel 6 into the AD_RESULT register; this effectively puts the command into the AD_COMMAND register to start the next conversion. Step 4 updates PTS_PTR1 (PTS_PTR1 points to 3004H), and step 5 leaves PTS COUNT at 7. The next cycle begins by storing the AD_COMMAND for Channel 5 in the ALU. During the 8th cycle (PTS COUNT at 1) the dummy command is loaded into the AD_COMMAND register, and no conversion is performed. PTS COUNT is decremented to zero and the end-of-PTS interrupt is requested. You can then right-shift by 6 bits all of the AD_RESULT data in the table to leave only the conversion results in these locations.

Table 12.2. A/D Scan Mode Table

Address	Contents
301EH	AD_RESULT for ACH0
301CH	0000 (Dummy command)
301AH	AD_RESULT for ACH1
3018H	AD_COMMAND for ACH0
3016H	AD_RESULT for ACH2
3014H	AD_COMMAND forACH1
3012H	AD_RESULT for ACH3
3010H	AD_COMMAND for ACH2
300EH	AD_RESULT for ACH4
300CH	AD_COMMAND for ACH3
300AH	AD_RESULT for ACH5
3008H	AD_COMMAND for ACH4
3006H	AD_RESULT for ACH6
3004H	AD_COMMAND for ACH5
3002H	AD_RESULT for ACH7
3000H	AD_COMMAND for ACH6

As another example, suppose you want to read the same A/D channel 10 times and store the result in the same location. This can be done with the A/D scan mode by setting SU = 0 to restore the original contents of PTS_PTR1 after the cycle. First, set up locations to hold the A/D command and result:

Address	Contents
3002H	XXXXH (will hold result)
3000H	AD_COMMAND for ACHx

Next, initialize PTS COUNT, PTS_CONTROL and the pointers:

Address	Contents
PTS_COUNT	0AH
PTS_CONTROL	0C3H (do not update SU)
PTS_PTR1	3000H
PTS_PTR2	1FAAH (the address of the SFR AD_RESULT)

User software starts a conversion on channel x. The PTS cycle begins when the conversion is finished. After the first cycle, the next conversion has been started and the result from the first conversion is in location 3002H. Before the second conversion is finished, the CPU could move the first result to a new location to make room for the second result. When the second cycle begins, PTS_PTR1 again points to 3000H to pick up the same command.

12.3.4 PTS Serial I/O Mode (SIO)

The PTS serial I/O (SIO) modes provide a software serial channel for the 8XC196MC. There are two modes ASIO (asynchronous serial I/O) and SSIO (synchronous serial I/O). At 16 MHz

the PTS can perform half duplex serial I/O at 9600 baud with only 4% CPU overhead. Up to 16 bits may be received or transmitted including the parity and stop bits.

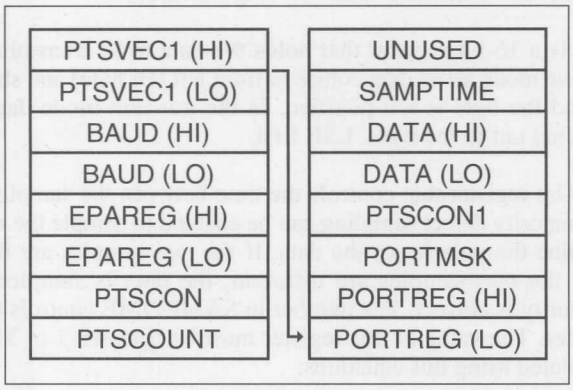


Figure 12.8. PTS SIO Control Block

The EPA provides the necessary timing and the input pins for receive. Any available output pin may be used for transmit.

The PTS has two control blocks for SIO operation, each containing eight 8-bit registers as shown in Figure 12.8.

PTSCOUNT controls how many PTS cycles will take place. This translates into how many bits are being received or transmitted in any SIO operation.

PTSCON selects various modes and types of SIO operation such as receive or transmit, synchronous or asynchronous, majority sample mode or no majority sample mode. The format of this byte is shown in Figure 12.9.

EPAREG (Lo and Hi) holds a 16-bit address for the compare or capture/compare time register used for timing of the SIO operation.

BAUD (Lo and Hi) holds a 16-bit value that controls the baud rate for SIO operation.

PTSVEC1 (Lo and Hi) is a 16-bit pointer for the second control block of PTS SIO.

PORTREG (Lo and Hi) holds the 16-bit address of the Px_PIN that contains the TXD or RXD.

PORTMSK defines which bit of Px_PIN will be used as the TXD/RXD pin. A 1 in a bit location defines the bit to be used for the special function. Only one bit can be specified for this function, all other bits must be 0.

PTSCON1 takes different bit patterns for synchronous and asynchronous modes. In the ASIO mode PTSCON1 enables parity, parity error flag and framing error flag (Figure 12.10). In the SSIO mode it has the receive/transmit select bit (Figure 12.16).

DATA (Lo and Hi) is a 16-bit register that holds the data to be transmitted or that has been received. In the receive mode serial data comes in from left (Hi byte) and shifted right. The LSB is shifted in first, and the byte is left justified. In the transmit mode data is loaded into the lower bits and is shifted out to the right, LSB first.

SAMPTIME is an 8-bit register that controls the time between the sampling of receiving bits. In the ASIO mode, majority detect sampling can be enabled to sample the received data at least two times to determine the polarity of the data. If the two samples are the same, the data is assumed correct. If the two samples are different, the data is sampled one more time to determine which polarity is correct. The number in SAMPTIME controls the number of states between these samples. The value of this register must be between 1 to 31. The time between samples can be calculated using this equations:

$$\text{Time between samples } (\mu s) = \frac{2 * (9 + \text{SAMPTIME})}{\text{fOSC (MHz)}}$$

This register is not used if majority sampling is disabled.

When cleared, bit 0 (MAJ) of PTSCON (Figure 12.9) enables majority sample mode. Bits 1 and 4 (SA), together determine the mode of operation. When both are 0, the asynchronous mode is selected, and when both are 1, the synchronous mode is selected. These two bits must always contain the same value.

PTSCON

7	6	5	4	3	2	1	0
M2	M1	M0	SA	0	0	SA	MAJ

M2, M1, M0	Mode	Function
	011	Transmit Mode
	001	Receive Mode
SA*	1	Synchronous
	0	Asychronous
Maj	1	Majority Sample enabled**
	0	Majority Sample disabled
0	These bits are reserved and must be 0	

*You must write the same value to both bits 1 and 4.
**Use only in Asynchronous Receive mode.

Figure 12.9. PTSCON for the ASIO and SSIO Modes

The transmit and receive functions are very similar in operation. An EPA module is programmed to provide the timing for data bits to be shifted in or out. Reloading the EPA

module and shifting the data bits are done in PTS cycles. EPA_REG holds the EPA time reload value for the next shift. PTSCOUNT keeps track of how many bits to be shifted in or out. At the end of each PTS cycle PTSCOUNT is decremented.

In the asynchronous receive mode an EPA pin is used so the start bit can be detected with use of the capture feature. In the synchronous modes an EPA output pin is used to emit the shift clock. Any compare module can keep track of the timing for SIO operation. The end of PTS interrupt service routine determines the next serial activity.

12.3.4.1 PTS Asynchronous Serial I/O Mode (ASIO)

To set up the PTS SIO in the asynchronous mode we need an EPA module to keep track of bit timing. We also need an I/O pin for a full duplex receive/transmit configuration. This I/O pin must correspond to the selected EPA module.

In a receive operation select a capture/compare module. The capture function is required in the receive mode to detect the start bit. In the transmit mode either a compare or a capture/compare module can be used for timing the bits. Any SFR output pin may be used in a half duplex transmit operation.

Compare modules generate the PTS requests at bit time intervals to scan the incoming bits. In the transmit mode a compare module generates one PTS request each bit time to shift the data out.

The following events take place every time a PTS cycle is requested. The EPA compare (or capture/compare) register located by EPAREG is read, the BAUD value is added to it, and the result is stored back to the EPA compare register. This sets up the next PTS interrupt. Next, the data bit is shifted into (receive) or out of (transmit) the DATA register from/to the corresponding port pin as defined by PORTREG and PORTMASK.

Finally, the PTSCOUNT register is decremented. If it is zero, the appropriate bit in PTS SRV is set to request an interrupt, and the corresponding PTS SEL bit is cleared to disable further PTS cycles for this channel.

In the majority sample mode, there are two interrupts set to occur when the EPA channel expires — one caused by the bit in the PTS SRV and one caused by the EPA module which has been programmed to occur at the next (un-needed) sampling time. The user interrupt routine should disable the EPA interrupt at the end of the serial operation.

The ASIO mode assumes that the last bit transmitted/received is always a stop bit with a value of 1. The last transmitted bit will always be a 1 independent of the value in the DATA register. During reception if a 0 is detected in the stop bit position, the FE (frame error) bit will set in the PTSCON1 register.

If parity mode is enabled (in receive or transmit), the second to last bit is assumed to be the parity bit. In transmit the calculated parity will be sent on the second to last bit. If the receive

parity does not match the calculated parity, the RPAR bit of PTSCON1 will indicate a parity error.

PTSCON1 ASIO MODE

7	6	5	4	3	2	1	0
0	RPAR	PEN	0	0	0	FE	TPAR

PEN: Parity Enable bit, 1 = enable, 0 = disable

FE: Framing Error flag, 1 = the stop bit received was not a 1.*, 0 = No Error. This bit must be reset at the start of every reception.

TPAR: Transmit Parity Control. This bit must be initialized at the start of every transmission. 1 = Odd, 0 = Even

RPAR: Receive Parity Control/Status. This bit has two functions; control and status. Before reception, initialize for Even (0) or Odd (1) parity. If at the end of a reception this bit is a 1, a parity error has occurred. This bit must be initialized before the start of every reception.

0: Write 0 to these bits.

Figure 12.10. PTSCON1 in ASIO Mode

The baud rate depends on three variables: oscillator frequency, contents of the BAUDCONST register and EPA prescale. Using the following equation you can find the decimal value that should be loaded to BAUD for a given baud rate.

BAUD = fOSC / (4 * baud rate * EPA prescale)

Figure 12.11. Asynchronous Baud Rate Equation

The following examples illustrates how to program the EPA and the PTS for asynchronous receive and transmit.

DATA REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0								

Figure 12.12. Format of the DATA Register 8-Bit Receive Mode

12.3.4.2 Example 1, Asynchronous Receive

In this example we will set up the PTS and the EPA to receive at 1200 baud. We will use P2.0 (CAPCOMP0) as the RXD pin, and set up to receive 1 start bit, 8 bits of data, no parity and 1 stop bit per byte. Figure 12.13 shows the bit pattern of a received frame and relative timing of the events that take place within that frame.

The EPA module CAPCOMP0 is programmed to capture on the falling edge of the start bit and generate an interrupt. In the interrupt service routine the user sets up the PTS to take over the bit recognition process. P2_PIN.0 is sampled during each PTS cycle (bit time) and the value is stored in DATA. At the end of each PTS cycle PTSCOUNT is decremented. After receiving the stop bit, the PTS count becomes 0 and generates an end of PTS interrupt. The interrupt service routine then stores the data and sets up for the next reception process.

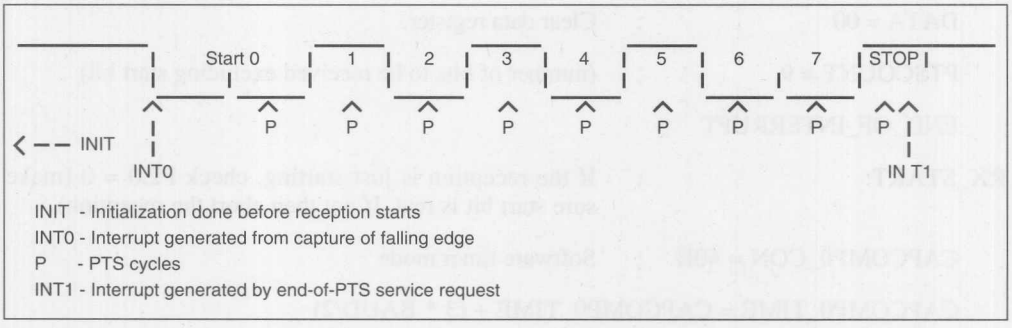


Figure 12.13. Receive Data Byte and Description of Terms

1. Configure P2.0 as an input port:
   - — P2_DIR.0 = 1 (select open-drain mode)
   - — P2_REG.0 = 1
   - — P2_MODE.0 = 0 (select port function)
2. Initialize PTS control registers:
   - — PTSCON = 20H (no majority sample mode)
   - — EPAREG = 1F42H (CAPCOMP0_TIME address)
   - — BAUD = 0D05H (from the baud rate formula)
   - — PTSVEC1 = pointer to PTSCB1
   - — PORTREG = 1FD6H (P2_PIN address)
   - — PORTMSK = 01H (bit 0 of P2_PIN selected as RXD)
   - — INT_MASK.2 = 1 (enable interrupt on CAPCOMP0)
3. Set up the EPA and the PTS for first reception:
   - — CAPCOMP0_CON = 10H (Capture on falling edge)
   - — PTSCON1 = 00 (clear control and status bits)
   - — DATA = 0000 (clear data register)
   - — PTSCOUNT = 9 (number of bits to be received: no start bit, 8 data bits, 1 stop bit.)
   - — EI, EPTS (re-enable interrupts/PTS)

A simplified interrupt routine for this function is outlined below:

```
CAPCOMP0_ISR;
    JBS CAPCOMP0_CON, 4, RX_START  ;  (Test for beginning/end of reception, this
                                      bit will set if interrupt was generated from
                                      edge detection).

RX_FINISH:                         ;  If the reception is finished, read data from DATA (HI)
                                      and read status from PTSCON1 (Check for framing
                                      error).
    CAPCOMP0_CON = 10H             ;  Set up next reception, falling edge detect.

    PTSCON1 = 00                   ;  Clear control/status register.

    DATA = 00                      ;  Clear data register.

    PTSCOUNT = 9                   ;  (number of bits to be received excluding start bit)

    END_OF_INTERRUPT

RX_START:                          ;  If the reception is just starting, check P2.0 = 0 (make
                                      sure start bit is real. If not then abort the reception).
    CAPCOMP0_CON = 40H             ;  Software timer mode

    CAPCOMP0_TIME = CAPCOMP0_TIME + (3 * BAUD/2)

                                   ;  Set up the software timer to occur 1.5 bit time after the
                                      falling edge of the start bit is detected.
                                      CAPCOMP0_TIME contains the time it captured at the
                                      falling edge of the start bit.
    PTS SEL.2 = 1                  ;  Enable the PTS as interrupt response.

    END_OF_INTERRUPT
```

12.3.4.3 Example 2, Asynchronous Transmit

The procedure to transmit in the ASIO mode is very much the same receive. In this mode we will not be capturing an external event, therefore, any EPA module can be used for timing. Also any SFR output port pin may be used as TXD pin.

In this example we will set up the PTS and the EPA to transmit at 1200 baud. We will use EPA module COMPARE0 as software timer, and its associated pin P2.4 as the TXD pin (although, that does not have to be an EPA pin). The set up will transmit 1 start bit, 8 bits of data, one parity (odd), and 1 stop bit per byte. TIMER0 with prescaler of 1 will be used as the time base for this operation.

DATA REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								D7	D6	D5	D4	D3	D2	D1	D0

Figure 12.14. Format of the DATA Register 8-bit Transmit Mode

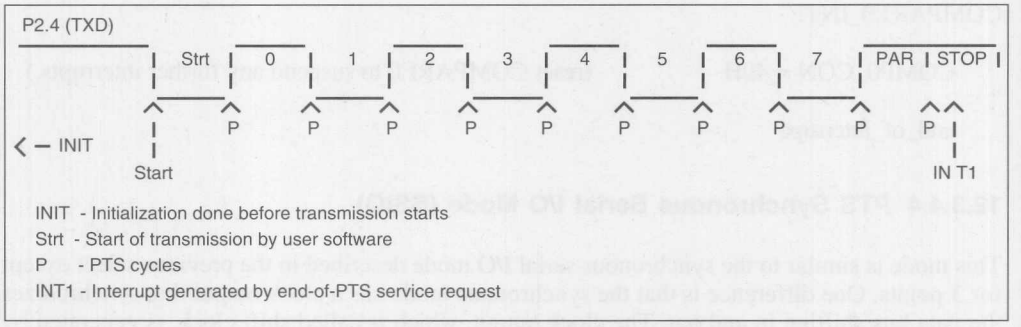


Figure 12.15. Asynchronous Transmit Data Waveform (DATA = 55H)

The following shows how to set up the PTS and the EPA for an asynchronous serial transmission channel:

1. Set-up P2.4 as an open-drain output
   - — P2_DIR.4 = 1 (Select open-drain mode)
   - — P2_REG.4 = 1
   - — P2_MODE.4 = 1 (Select port function)
2. Initialize PTS Control Blocks / PTS registers / Misc.
   - — PTSCON = 60H (ASIO Transmit Mode)
   - — EPAREG = 1F5AH (COMP0_TIME address)
   - — BAUD = 3333 (from the equation on Figure 12.11)
   - — PTSVEC1 = pointer to PTSCB1
   - — PORTREG = 1FD4H (P2_REG address)
   - — PORTMSK = 10H (bit 4 of P2_PIN designated as TXD)
   - — INT_MASK.3 = 1 (enable interrupt on COMPARE0)
   - — COMP0_CON = 40H (Configure module as software timer in this PTS mode)
3. Start first transmission
   - — PTSCON1 = 21h (parity enable, odd)
   - — DATA (Lo) = XXH (load low byte of data register with 8-bit data to be transmitted)
   - — PTSCOUNT = 10 (number of bits to be transmitted, excluding start, including parity/stop)
   - — DI, DPTS (interrupt/PTS protect the start bit and read of timer)
   - — P2_REG.4 = 0 (start bit)

— COMP0_TIME = TIMER1 + BAUD (set-up time for 1st PTS Cycle)
— PTS SEL.3 = 1 (enable PTS response for interrupt requests)
— EI, EPTS (re-enable interrupts/PTS)

4. The PTS will transmit the remaining bits transparent to the user. To start another transmission, repeat step 3.

This interrupt routine is entered at once at the end of a transmission,

```
COMPARE0_INT:

    COMP0_CON = 40H          (reset COMPARE0 to suspend any further interrupts.)

    end_of_interrupt
```

12.3.4.4 PTS Synchronous Serial I/O Mode (SSIO)

This mode is similar to the synchronous serial I/O mode described in the previous select except for 3 points. One difference is that the synchronous mode has a clock output that synchronizes the data bits shifting in and out. The clock output, which is called shift clock, is generated by the EPA and the PTS.

Each data bit is synchronized with the shift clock. There are two transitions of shift clock per data bit (see Figure 12.18). Therefore, we need 16 PTS cycles to receive or transmit 8 bits of data. Every PTS cycle toggles the shift clock and every other PTS cycle shifts the data in or out.

Another difference is that only data bits are shifted in or out. There are no parity or stop bits included in a received or transmitted frame. Finally, the majority sampling mode is not supported in this mode.

PTSCON1 in SSIO Mode

7	6	5	4	3	2	1	0
0	0	0	0	0	0	TRC	0

TRC: Transmit/Receive Control

1 = Receive or Transmit data bit on first PTS request and every other one thereafter

0 = Receive or Transmit data bit on second PTS request and every other one thereafter. This bit must be re-initialized at the start of every transmission or reception. Throughout the transmission/reception, this bit is toggled by hardware.

0: Reserved, write 0 to these bits.

Figure 12.16. PTSCON1 in SSIO Mode

To distinguish between the PTS cycles that toggle the shift clock and those that shift a data bit, a flag is provided in PTSCON1. Bit 1 of this register is called TRC (transmit receive

control). The user sets or clears this bit once in the PTS set up section of the code, and it automatically is toggled every PTS cycle thereafter.

If TRC is 1, the PTS cycle toggles the shift clock, and shifts the data bit in or out of the DATA register. If TRC is 0, the PTS cycle only toggles the shift clock. The TRC bit is toggled at every PTS cycle to enable and disable data shifting.

The TRC bit also determines which edge of the shift clock the data bits are synchronized with. In the receive mode data bits may be sampled at rising or falling edge of the shift clock. In the transmit mode data bits may be shifted out on rising or falling edge of the shift clock.

In the following examples, TRC is cleared in the receive mode and set in the transmit mode. Clearing TRC corresponds to sampling of the data bit on the falling edge of the shift clock. Setting TRC corresponds to shifting of the data bit on the rising edge of the shift clock.

Events In a PTS Cycle. The following events take place every time a PTS cycle is requested. The EPA compare or capture/compare register (COMPx_CON or CAPCOMPx_CON) located by EPAREG is read, the BAUD value is added to it, and the result is stored back to the EPA compare register. This sets up the next PTS interrupt.

The data is shifted to or from the port pin that is located by PORTREG and PORTMSK. At the end of each PTS cycle, the PTSCOUNT register is decremented. When it becomes zero, the appropriate PTS SRV bit is set to request an interrupt, and the corresponding PTS SEL bit is cleared to disable further PTS cycles.

In the synchronous mode the baud rate depends on the oscillator frequency, EPA prescale, and the value of BAUD. To determine the value of BAUDCONST use the following equation.

BAUD = fOSC / (8 * baud rate * EPA prescale)

Figure 12.17. Asynchronous Baud Rate Equation

12.3.4.5 Example 3. Synchronous Receive Mode

The following example shows the steps to set up the EPA and the PTS for a synchronous receive operation. 8-bits of data (54H) will be received at the rate of 1200 baud. EPA Timer1 is free-running with prescale of 1. CAPCOMP0 provides the timing for the shift clock and puts it out on P2.0. Any SFR I/O pin may be used for data. In this example P2.4 is used as an input pin. Because this is a synchronous operation, timing that CAPCOMP0 provides for shift clock is also used for data bits.

Data is sampled and received on falling edge of the shift clock.

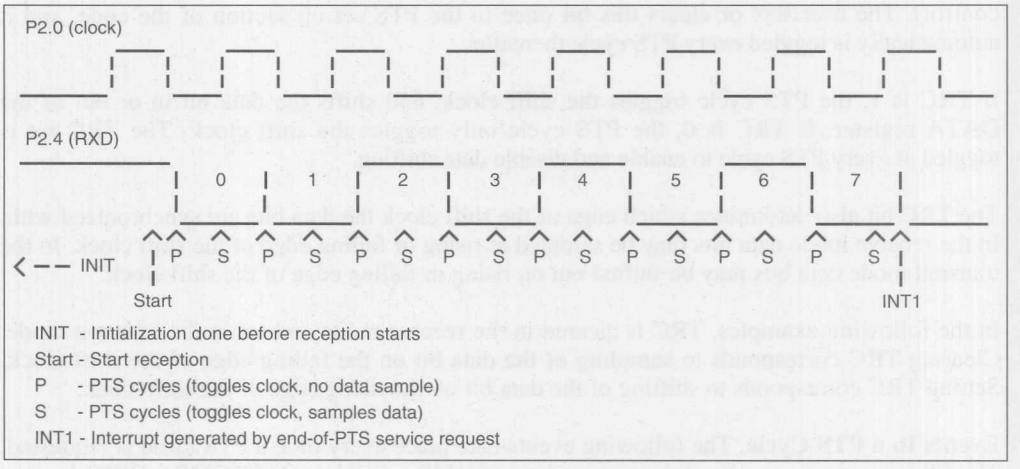


Figure 12.18. Synchronous Receive

The following shows the required steps to set up the PTS and the EPA for SSIO in receive mode.

1. Set-up P2.4 as an input
   - — P2_DIR.4 = 1 (Select open-drain mode)
   - — P2_REG.4 = 1
   - — P2_MODE.4 = 0 (Select port function)
2. Set-up P2.0 as CAPCOMP0 pin (output)
   - — P2_DIR.0 = 0 (configure pin as output)
   - — P2_REG.0 = 0 (Initialize clock to 0)
   - — P2_MODE.0 = 1 (Select EPA output function)
3. Initialize PTS Control Blocks, PTS registers, Misc.
   - — PTSCON = 32H (SSIO Receive Mode)
   - — EPAREG = 1F42H (CAPCOMP0_TIME address)
   - — BAUD = 0683H
   - — PTSVEC1 = POINTER TO PTSCB1
   - — PORTREG = 1FD7H (P2_PIN address)
   - — PORTMSK = 10H (bit 4 of P2_PIN selected as RXD)
   - — INT_MASK.2 = 1 (enable interrupt on CAPCOMP0)
   - — CAPCOMP0_CON = 70H (interrupt and toggle pin)
4. Start reception
   - — PTSCON1 = 00 (sample data on second PTS request)
   - — DATA = 00 (clear data register)
   - — PTSCOUNT = 16 (number of bits to be received × 2 for SSIO modes)
   - — PTS SEL.2 = 1 (enables PTS response for interrupt requests)
   - — CAPCOMP0_TIME = TIMER1 + 10 (set-up first PTS immediately)

5. The PTS cycles will receive the data and at the end will generate an end of PTS interrupt. If more data is to be received, repeat steps 3 and 4.

This interrupt routine is entered once at the end of a SSIO reception.

```
CAPCOMP0_INT:
    CAPCOMP0_CON = 70H        (reset CAPCOMP0 value, cancels and possible
                               interrupts)

    Read Data from DATA (Hi)  (Data received was 8 bit, no shift of DATA necessary.
                               Read high byte directly)

    end_of_interrupt
```

12.3.4.6 Example 4, Synchronous Transmit Mode

The following example shows the steps to set up the EPA and the PTS for a SSIO operation. 8-bits of data will be transmitted at the rate of 1200 baud. EPA Timer1 is free-running with prescale of 1. CAPCOMP0 provides the timing for the shift clock and puts it out on P2.0. Any SFR I/O pin may be used for data. In this example P2.1 is used as an output pin. Because this is a synchronous operation, timing that CAPCOMP0 provides for shift clock is also used for data bits.

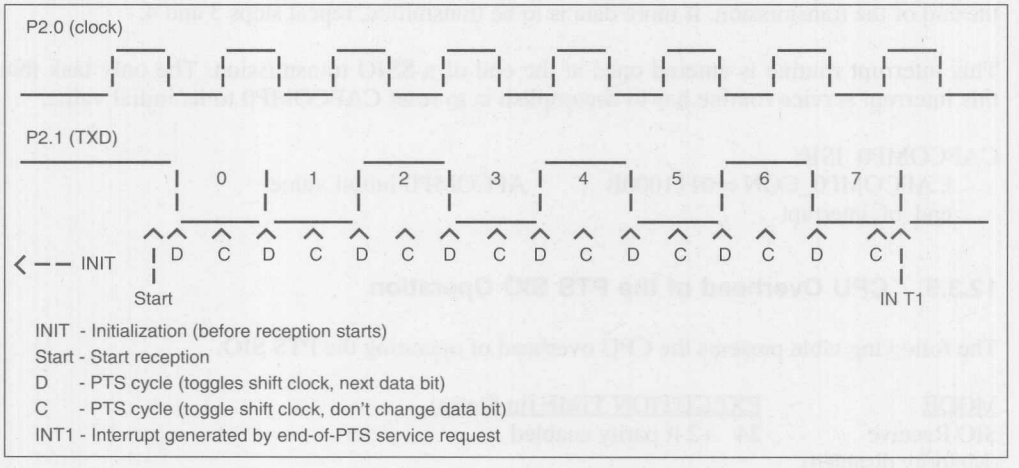


Figure 12.19. Synchronous Serial Receive Mode

1. Set-up P2.1 as an output for data channel
   — P2_DIR.1 = 1 (select open drain)
   — P2_REG.1 = X (Output value is a don't care)
   — P2_MODE.1 = 0 (Select port function as an output)

2. Set-up P2.0 as CAPCOMP0 pin (output) for shift clock
   — P2_DIR.0 = 0 (select complementary output)
   — P2_REG.0 = 0 (Initialize clock to 0)
   — P2_MODE.0 = 1 (Select EPA output function)

3. Initialize PTS Control Blocks and Registers
   — PTSCON = 01110010B (SSIO Transmit Mode)
   — EPAREG = 1F42H (CAPCOMP0_TIME address)
   — BAUD = 0683H (for a 16MHz oscillator frequency)
   — PTSVEC1 = pointer to PTSCB1
   — PORTREG = 1FD4H (P2_REG address)
   — PORTMSK = 02 (bit 1 of P2_PIN selected as TXD)
   — INT_MASK.2 = 1 (enable interrupt on CAPCOMP0)
   — CAPCOMP0_CON = 0111000B (toggle output pin as shift clock, enable EPA interrupt)

4. Start transmission
   — PTSCON1 = 02H (change data on first PTS request)
   — DATA1 = XX (data byte to be transmitted)
   — PTSCOUNT = 16 (number of bits to be received × 2 for SSIO modes)
   — PTS SEL.2 = 1 (enable PTS response for interrupt requests)
   — CAPCOMP0_TIME = TIMER1 + 10 (set-up first PTS immediately)

The PTS cycles will generate the shift clock and transmit the data. An interrupt will occur at the end of the transmission. If more data is to be transmitted, repeat steps 3 and 4.

This interrupt routine is entered once at the end of a SSIO transmission. The only task that this interrupt service routine has to accomplish is to reset CAPCOMP0 to its initial value.

```
CAPCOMP0_ISR:
    CAPCOMP0_CON = 0111000B  ;      APCOMP0 initial value
    end_of_interrupt
```

12.3.5 CPU Overhead of the PTS SIO Operation

The following table presents the CPU overhead of operating the PTS SIO.

MODE	EXECUTION TIME (in States)
SIO Receive (Majority disabled)	24 +2 if parity enabled
SIO Receive (Majority enabled)	36 +(sample time) for second sample +(7 + sample time) for third sample +2 if parity enabled

NOTE

The samples in this mode are taken an equal distance apart, although the above table seems to indicate otherwise. The number of states between the samples is (9 + value of sample time).

SIO Transmit	29	+3 if parity enabled
SSIO Receive	29	(receive data bit)
	21	(no reception)
SSIO Transmit	30	(transmit data bit)
	20	(no transmission)

Special Modes of Operation

13

CHAPTER 13
SPECIAL MODES OF OPERATION

The 8XC196MC has idle and powerdown modes to reduce the amount of current consumed by the chip. The ONCE (ON-Circuit-Emulation) mode isolates the 8XC196MC from the other system components.

13.1 IDLE MODE

In the idle mode the CPU stops executing. The CPU clocks are frozen at logic state zero, but peripheral clocks and CLKOUT continue active. Power consumption in the idle mode is reduced to about 40% of the active mode power. The idle mode is entered by executing the instruction ''IDLPD #1''.

In the idle mode, the system bus control pins (ALE, $\overline{RD}$, $\overline{WR}$, INST and $\overline{BHE}$), go to their inactive states. If ports 3 and 4 are configured as I/O ports, the pins retain the current values in their data latches. If these ports are configured for the ADDR/DATA bus, the pins float.

NOTE

The watchdog timer continues to run in the idle mode if it is enabled. (The watchdog is enabled if WD = 0 or if WD = 1 and WATCHDOG has been cleared. WD = CCB1.3; see Section 14.5.2) In this case the chip will reset unless it is awakened every 64K state times to clear WATCHDOG.

An enabled interrupt or a hardware reset returns the CPU from idle mode to its active state. As all of the peripherals are running in idle mode, the interrupt can be generated by the EPA, ASIO, SSIO or A/D, as well as NMI, or EXTINT. When an interrupt brings the CPU out of the idle mode, the CPU vectors to the corresponding interrupt service routine and begins executing. The CPU returns from the interrupt service routine to the next instruction following the ''IDLPD #1'' instruction that put the CPU in idle mode.

13.2 POWERDOWN MODE

In powerdown mode, all internal clocks are frozen at logic state zero and the oscillator is shut off. The register file, internal ram, and most peripherals hold their values if V_{CC} is maintained. Power is reduced to the device leakage and is in the μA range. Powerdown mode is entered by executing the instruction, ''IDLPD #2''.

In powerdown, the bus control pins go to their inactive states. All of the output pins assume the values in their data latches. If ports 3 and 4 are configured as I/O ports, they retain the current values in their data latches. If these ports are configured for the ADDR/DATA bus, the pins float.

If the PD bit of the chip configuration bytes (CCB0.0) is cleared, then the powerdown mode is disabled. Having PD = 0 in ROM/EPROM prevents accidental entry into powerdown.

Powerdown mode can be exited by a chip reset or by a rising edge on the external interrupt (EXTINT) pin. If the $\overline{\text{RESET}}$ pin is used, it must be asserted long enough for the oscillator to stabilize. If EXTINT is used, an internal timing circuit ensures that the oscillator has time to stabilize before the internal clocks are turned on. An external capacitor on the V_{PP} pin determines the time constant. In powerdown, a rising edge on EXTINT generates an interrupt regardless of the value of the EXTINT mask bit.

Figure 13.1 shows the power down and power up sequence using an external interrupt. During normal operation (before entering powerdown mode) the V_{PP} pin is held at V_{CC} through an internal pull-up. The user must connect a capacitor between V_{PP} and V_{SS}. A positive level on the external interrupt pin starts to discharge this capacitor. The internal current source that discharges the capacitor can sink approximately 100 μA. When the voltage goes below about 1V on the V_{PP} pin, the chip begins executing code. (A 1μF capacitor would take about 4 ms to discharge to 1V.) V_{PP} then begins to rise back to V_{CC}.

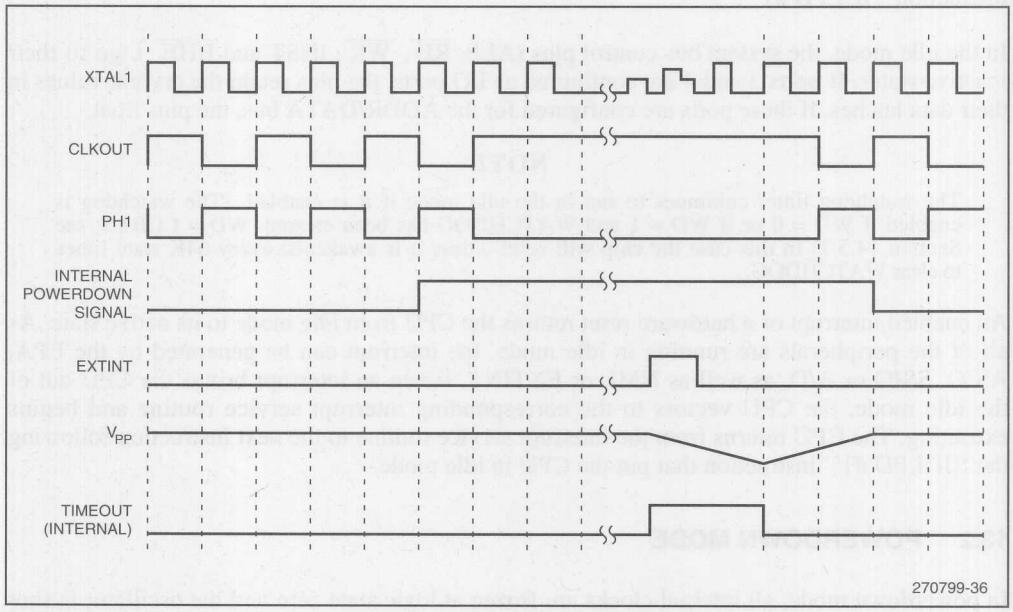


Figure 13.1. Power Down and Power Up Sequences

If the external interrupt brings the chip out of powerdown, the corresponding bit is set in the interrupt pending register. If the interrupt is unmasked, the device immediately executes the interrupt service routine. After the interrupt service routine is finished, control returns to the instruction following the IDLPD instruction that put the chip into powerdown. If the interrupt is masked, the chip starts at the instruction following the IDLPD instruction. The bit in the pending register remains set, however.

All peripherals should be in an inactive state before entering powerdown. An incomplete A/D conversion is aborted. If the chip comes out of powerdown via the external interrupt, the serial port continues where it left off. Make sure that the serial port is finished transmitting or receiving before entering powerdown. The SFRs associated with the A/D and the serial port may contain incorrect information when returning from powerdown.

When the chip is in powerdown, it is impossible for the watchdog timer (Section 14.5.2) to time out because its clock has stopped. If you are using powerdown and the watchdog timer (watchdog enable bit, WD = CCB1.3 = 0), you should clear WATCHDOG just before entering powerdown. This prevents the watchdog from timing out when the oscillator is stabilizing after leaving powerdown. If the watchdog is inactive (WD = 1 and WATCHDOG has not been cleared since reset), *do not* clear WATCHDOG, as this would activate the watchdog.

13.3 ONCE AND OTHER TEST MODES

The only test mode not reserved for use by Intel is the ONCE (ON-Circuit-Emulation) Mode. The ONCE Mode is useful for electrically removing the 8XC196MC from the rest of the system. A typical application of the ONCE Mode is to program discrete EPROMs on the circuit board without removing the 8XC196MC from its socket.

ONCE is entered by holding P5.4 low on the rising edge of $\overline{\text{RESET}}$. During ONCE, all pins except XTAL1 and XTAL2 float. $\overline{\text{RESET}}$ must remain high during ONCE. If $\overline{\text{RESET}}$ is brought low during ONCE the reset mode will be entered, and the pins will no longer float; they will be weakly pulled high or low (see section 14.5).

Test modes are entered if P2.6 or P5.4 is held low on the rising edge of $\overline{\text{RESET}}$. For this reason it is crucial to meet the specification for I_{OH} in Reset. The P5.4 and P2.6 source about 50 µA at a logical 1 during reset. Consult the data sheet for the exact specification.

To ensure that the user does not mistakenly enter a Test Mode, *do not use pins P2.6 and P5.4 as inputs.*

Minimum Hardware Considerations

CHAPTER 14
MINIMUM HARDWARE CONSIDERATIONS

The 8XC196MC requires several external conditions to operate correctly. Power and ground must be connected, a clock signal must be generated, and a reset circuit must be present. Sections 14.1 through 14.3 describe power and ground connections and methods to avoid noise problems. Section 14.4 discusses the external oscillator and the internal timing signals derived from it. The reset sequence, its initiation, and its consequences are described in Section 14.5.

External Signals.

V_{CC}	Supply voltage
V_{SS} (3 pins)	Ground
V_{REF}	Reference voltage for A/D converter
ANGND	Analog ground for A/D converter
XTAL1	Crystal/resonator or external clock input
XTAL2	Inverted output for crystal/resonator
$\overline{\text{RESET}}$	Reset signal
V_{PP}	EPROM Programming supply voltage

14.1 POWER SUPPLY PINS

Power to the 8XC196MC flows through 7 to 9 (depending on the package) pins (omitting V_{PP}, the supply for EPROM programming, discussed in Section 16). V_{CC} supplies the positive voltage to the digital portion of the chip, while V_{REF} supplies the A/D converter and PORT 0 and PORT 1with a positive voltage. These two pins must be connected to 5V power supplies. When the A/D converter is in use, it is desirable to connect V_{REF} to a separate supply, or at least a separate trace, to minimize noise in the A/D converter. V_{REF} and ANGND must be connected for PORT 0 to function even if the A/D converter is not being used. Figure 14.1 shows connections for V_{SS}, V_{CC}, V_{REF} and ANGND.

14.2 COMMON RETURN PINS

The common return pins (V_{SS} and ANGND) must all be nominally at 0 V. Figure 14.1 shows their connections. For best A/D operation it is important to isolate the V_{SS} lines from ANGND. The V_{SS} lines should be connected by the shortest possible path and decoupled to V_{CC} at each pin. If the board has separate V_{CC} and ground planes, connect ANGND to the digital ground plate at a single point near the 8XC196MC. If ground traces are used, ANGND should be connected to V_{SS} at a single point near the 8XC196MC. Even if the A/D is not used, V_{REF} and ANGND must be connected for Port 0 to function.

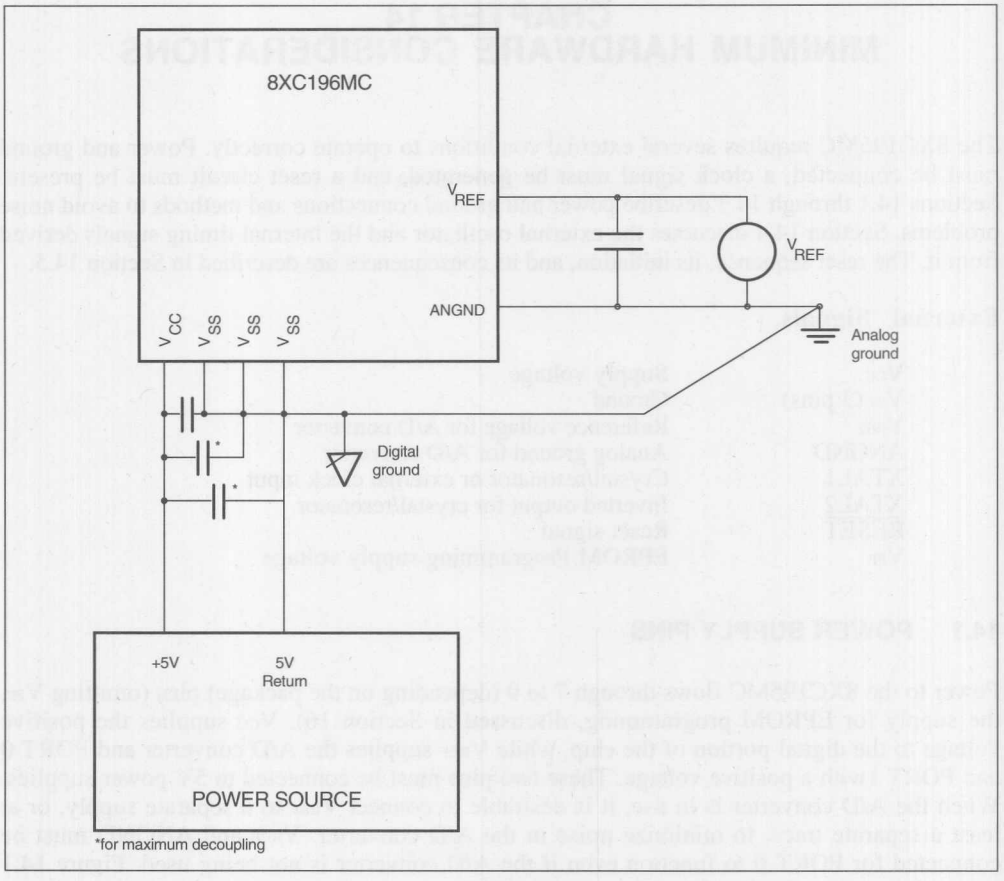


Figure 14.1. Power and Return Connections

14.3 NOISE PROTECTION TIPS

Due to the fast rise and fall times of high speed CMOS logic, noise spikes on the power supply lines and outputs at the chip are not uncommon. It is, therefore, extremely important to follow good design and board layout techniques to minimize noise. Liberal use of decoupling capacitors, transient absorbers, and V_{CC} and ground planes can help greatly. For maximum protection, decoupling capacitors should be used at all common return pins. It is much easier to design a PC board with these features than to search for random noise on a poorly designed board. For more information on noise, refer to Application Note AP-125, ''Designing Microcontroller Systems for Noisy Environments.''

14.4 OSCILLATORS AND INTERNAL TIMINGS

14.4.1 On-Chip Oscillator

The on-chip oscillator circuitry shown in Figure 14.2, consists of a crystal-controlled, positive reactance oscillator. In this application, the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

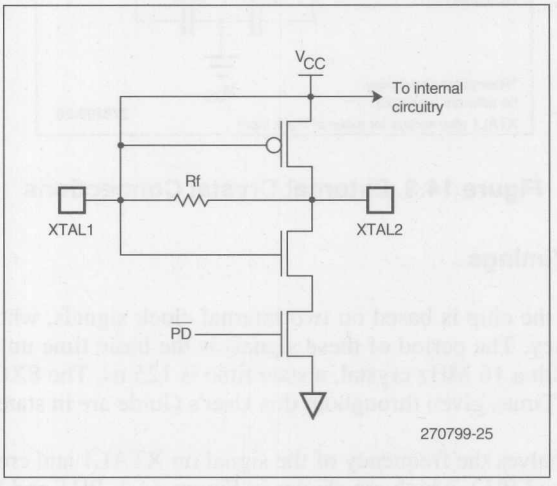


Figure 14.2. On-Chip Oscillator Circuitry

The feedback resister, R_f, consists of paralleled n-channel and p-channel FETs. Operation of the oscillator is controlled by the PD (powerdown) bit in the IDLPD instruction. In the powerdown mode (Section 13.2) R_f acts as an open circuit and shuts off the oscillator. The XTAL1 and XTAL2 pins also have electrostatic discharge (ESD) protection, which is not shown in the figure.

The capacitance values in Figure 14.3 are not critical. 20 pF is adequate for any frequency above 3.5 MHz with good quality crystals. A ceramic resonator can be used instead of a crystal in cost sensitive applications. However, the external circuit required with a resonator may be different from that for a crystal. Contact the resonator manufacturer for appropriate resistor and capacitor values.

Noise spikes arriving at the XTAL1 or XTAL2 pins can cause a miscount in the internal clock-generating circuitry. These spikes can be introduced via capacitive coupling between the oscillator components and circuit board traces carrying digital signals with fast rise times. For this reason, the oscillator components should be mounted close to the chip and should have short, direct traces to XTAL1, XTAL2 and V_{SS}.

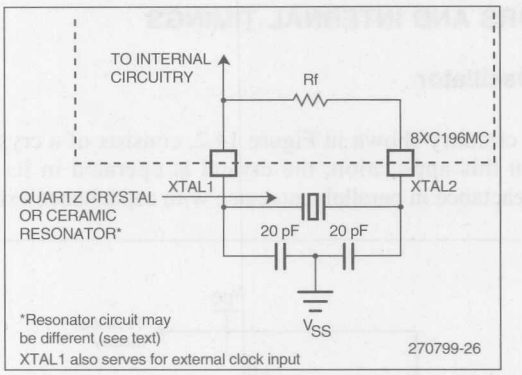


Figure 14.3. External Crystal Connections

14.4.2 Internal Timings

Internal operation of the chip is based on two internal clock signals, whose frequency is half the oscillator frequency. The period of these signals is the basic time unit, known as a ''state time'' or ''state.'' With a 16 MHz crystal, a state time is 125 ns. The 8XC196MC can operate at many frequencies. Times given throughout this User's Guide are in state times.

The clock generator halves the frequency of the signal on XTAL1 and creates the two internal timing signals, PH1 and PH2, which are shown in Figure 14.4. PH1 and PH2 are active when high. The non-overlapping active portions of PH1 and PH2 define two time periods, *Phase 1* and *Phase 2*. CLKOUT is generated by the rising edges of PH1 and PH2. This is not the same as in the 8096BH, which uses a three-phase clock. Changing from a three-phase clock to a two-phase clock speeds up operation for a set oscillator frequency. In comparison with the 8096BH, the internal bus rate of the 8XC196MC is 33% faster at the same frequency. AC timing parameters are described in Section 15.4. Consult the latest data sheet for AC timing specifications.

14.5 $\overline{\text{RESET}}$ AND RESET STATUS

Reset initializes the 8XC196MC to a known state. During the reset sequence the chip configuration bytes (CCBs, Section 15.2) are read from locations 2018H and 201AH and stored in the chip configuration registers (CCRs). According to the state of the external access signal, the CCBs are read from external ($\overline{\text{EA}}$ = 0) or internal ($\overline{\text{EA}}$ = 1) memory. Table 14.1 gives the status of all of the pins after reset, as well as in the idle and powerdown (PD) modes (Section 13). The pins for Ports 2 and 5 have weak pull-ups that force each port pin to a 1 at reset. Table 14.2 lists the states of the special function registers (SFRs) after the reset sequence.

The 8XC196MC can be reset in four ways: by an external signal on the $\overline{\text{RESET}}$ pin, by the watchdog timer, with the RST instruction, and by an idle/powerdown (IDLPD) instruction with illegal keys (other than 0 or 1).

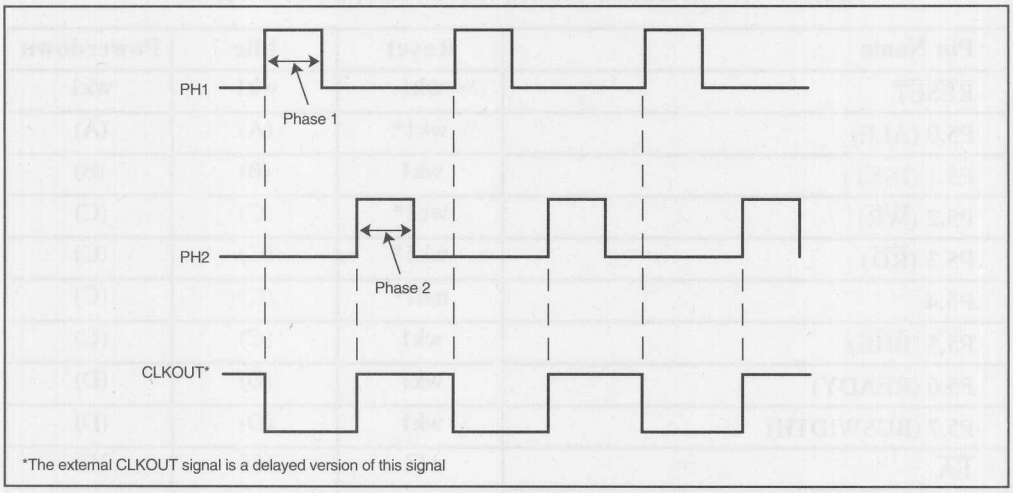


Figure 14.4. Internal Clock Phase

14.5.1 $\overline{\text{RESET}}$ and Reset Circuits

The simplest way to reset the 8XC196MC is to insert a capacitor between the $\overline{\text{RESET}}$ pin and V_{SS}. The 8XC196MC has an internal pull-up with a value between 6 KΩ and 65 KΩ. A capacitor of 5 μF or greater should provide sufficient reset time as long as V_{CC} rises quickly. If a spike on V_{CC} or the $\overline{\text{RESET}}$ line causes $\overline{\text{RESET}}$ to be pulled low, then $\overline{\text{RESET}}$ will remain low for 16 state times. Also, any external reset pulses less than 16 states will be stretched for the remainder of the 16 states. After the $\overline{\text{RESET}}$ pin is returned high, the reset sequence occurs as shown in Figure 14.5.

Figure 14.6 shows the internal structure of the $\overline{\text{RESET}}$ pin. This pin functions as an input to reset the 8XC196MC and also as an output to reset an entire system in response to a WATCHDOG overflow, a RST instruction execution, or execution of illegal Idle/Powerdown instruction keys. For a system reset application, the reset circuit should be a one-shot with an open collector output. The reset pulse may have to be lengthened and buffered because $\overline{\text{RESET}}$ is asserted for only 16 state times. A capacitor cannot be connected directly to $\overline{\text{RESET}}$ if it is to drive the reset pins of other chips in the circuit. The capacitor may keep the voltage on the pin from going below guaranteed V_{IL} for circuits connected to the $\overline{\text{RESET}}$ pin. Figure 14.7 shows an example of a system reset circuit.

Table 14.1. Pin States during Reset, Idle, and Powerdown

Pin Name	Reset	Idle	Powerdown
$\overline{\text{RESET}}$	wk1	wk1	wk1
P5.0 (ALE)	wk1*	(A)	(A)
P5.1 (INST)	wk1	(B)	(B)
P5.2 ($\overline{\text{WR}}$)	wk1*	(C)	(C)
P5.3 ($\overline{\text{RD}}$)	wk1*	(C)	(C)
P5.4	md1*	(C)	(C)
P5.5 ($\overline{\text{BHE}}$)	wk1	(C)	(C)
P5.6 (READY)	wk1	(D)	(D)
P5.7 (BUSWIDTH)	wk1	(D)	(D)
$\overline{\text{EA}}$	HZ	HZ	HZ
NMI	wk0	wk0	wk0
P3,P4 (EA = 0)	wk1	HZ	HZ
P3,P4 (EA = 1)	wk1	ODIO	ODIO
CLKOUT	clk	clk	0,LZ
EXTINT	HZ	HZ	HZ
P0 (ACH)	HZ	HZ	HZ
P1 (ACH)	HZ	HZ	HZ
P2.0	wk1*	(E)	(E)
P2.[7,5:1]	wk1	(E)	(E)
P2.6	md1*	(E)	(E)
P6.[5:0]	wk1	(F)	(F)
P6.[7:6]	wk0	(F)	(F)
V_{PP}	HZ	1,LZ	1,LZ
XTAL1	HZ	HZ	HZ
XTAL2	osc,LZ	osc,LZ	(G)

NOTES: (A) if P5_MODE.0 = 0 then port value
if P5_MODE.0 = 1 and CCR.3 = 1 (ALE mode) then LZ 0
if P5_MODE.0 = 1 and CCR.3 = 0 (ADV mode) then LZ 1

(B) if P5_MODE.1 = 0 then port value
if P5_MODE.1 = 1 then LZ 0

(C) if P5_MODE.x = 0 then port value
if P5_MODE.x = 1 then LZ 1

(D) if P5_MODE.x = 0 then port value

if P5_MODE.x = 1 then HZ

(E) if P2_MODE.x = 0 then port value
if P2_MODE.x = 1 then as peripheral specifies

(F) if output port then port value
if special function then as peripheral specifies

(G) if XTAL1 = 1 then LZ 0
if XTAL1 = 0 then LZ 1

HZ – High Impedance
IZ – Low Impedance
wk1 – Weakly Pulled High
wk0 – Weakly Pulled Low
md1 – Medium Strength High
ODIO – Open Drain IO
clk – Clock

*These pins are also used to control test mode entry.

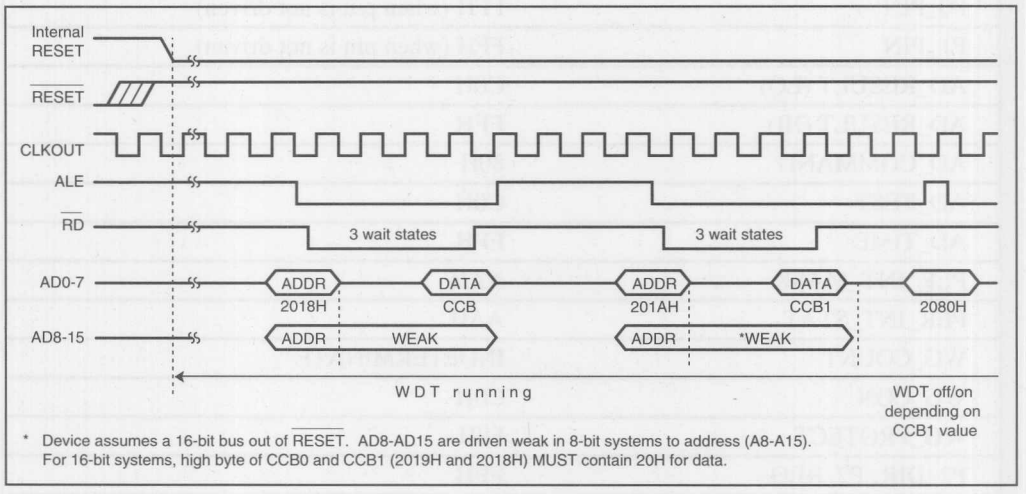


Figure 14.5. Reset Sequence

14.5.2 Watchdog Timer (WATCHDOG)

The state of the watchdog timer is determined by WD, the watchdog enable bit (CCB1.3). If WDD = 1, the watchdog is inactive; but it can be activated by clearing WATCHDOG. To clear WATCHDOG, write 1EH to location 0AH and then immediately write 0E1H to 0AH. Following a clear, the watchdog will reset the 8XC196MC after 65535 state times. If WD = 0, the watchdog is active and will reset the 8XC196MC after the 65535 state times unless it is periodically cleared.

Table 14.2. SFR Reset Values

SFR	RESET VALUE
CAPCOMP0_TIME	INDETERMINATE
CAPCOMP1_TIME	INDETERMINATE
CAPCOMP2_TIME	INDETERMINATE
CAPCOMP3_TIME	INDETERMINATE
COMPARE0_TIME	INDETERMINATE
COMPARE1_TIME	INDETERMINATE
COMPARE2_TIME	INDETERMINATE
COMPARE3_TIME	INDETERMINATE
T1RELOAD	INDETERMINATE
P0_PIN	FFH (when pin is not driven)
P1_PIN	FFH (when pin is not driven)
AD_RESULT (LO)	C0H
AD_RESULT (HI)	FFH
AD_COMMAND	80H
AD_TEST	C0H
AD_TIME	FFH
PER_INT_MASK	AAH
PER_INT_STAT	AAH
WG_COUNT	INDETERMINATE
WG_CON	C0H
WG_PROTECT	F0H
P2_DIR, P2_REG	FFH
P2_PIN	FFH (when pin is not driven)
P5_MODE	80H IF $\overline{EA}$ = HIGH, A9H IF $\overline{EA}$ = LOW
P5_DIR, P5_PIN	FFH
P5_REG	FFH (when pin is not driven)
USFR	02H
P3_REG, P4_REG	FFH
P3_PIN, P4_PIN	FFH (when pin is not driven)

NOTE: This table lists all the registers that their reset value is not 0. Given values include the reserved bits (when applicable).

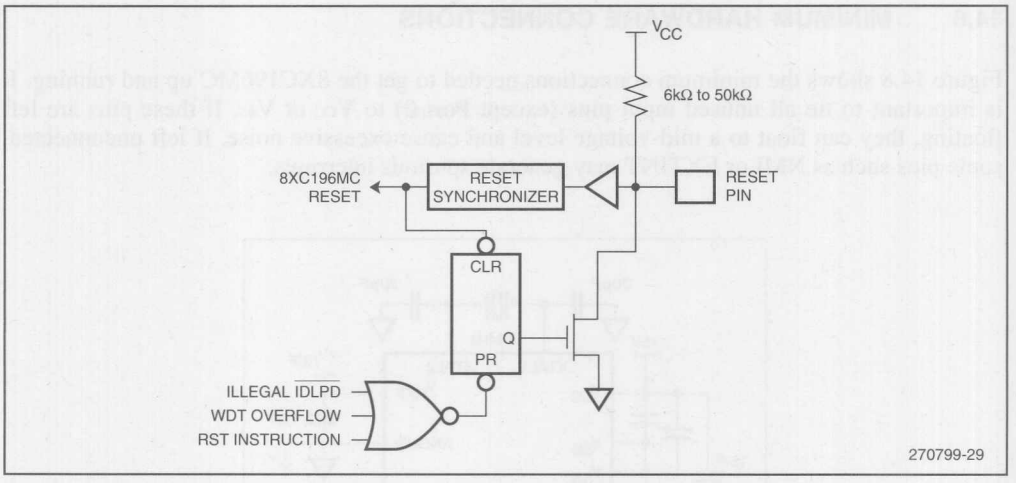


Figure 14.6. Reset Pin

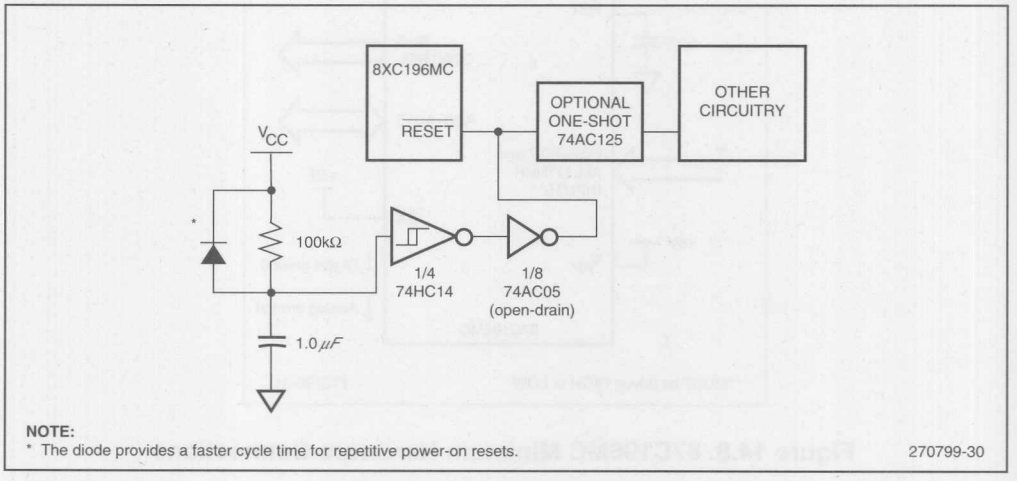


Figure 14.7. System Reset Circuit

14.5.3 RST Instruction

Executing a RST instruction also resets the 8XC196MC. The RST opcode is 0FFH. Putting pull-ups on the address/data bus causes unimplemented areas of memory to be read as 0FFH. If unused internal EPROM or ROM memory is set to 0FFH, then execution from unimplemented or unused memory will reset the 8XC196MC.

14.6 MINIMUM HARDWARE CONNECTIONS

Figure 14.8 shows the minimum connections needed to get the 8XC196MC up and running. It is important to tie all unused input pins (except Port 0) to V_{CC} or V_{SS}. If these pins are left floating, they can float to a mid-voltage level and cause excessive noise. If left unconnected, some pins such as NMI or EXTINT may generate spurious interrupts.

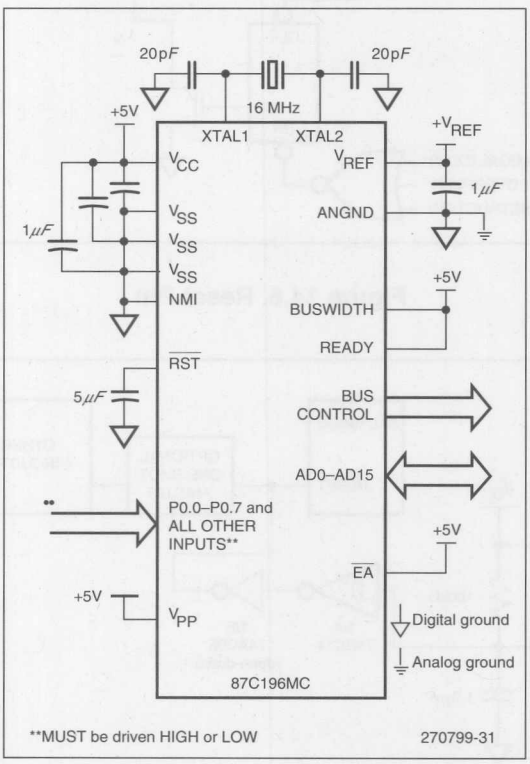


Figure 14.8. 87C196MC Minimum Hardware Connections

External Memory Interfacing

15

CHAPTER 15
EXTERNAL MEMORY INTERFACING

The 8XC196MC can be interfaced with external memory in a variety of configurations. The user sets the chip configuration registers to specify the features required for a particular external memory design. These features include 8- and/or 16-bit bus mode selection, wait state control for slow external devices, ROM/EPROM protection options and the selection of multiplexed external signals.

External Memory Signals. The following external signals are described in Section 15.

$\overline{EA}$		External memory access
ALE/$\overline{ADV}$	P5.0	Address latch enable/address valid
INST	P5.1	Instruction fetch from external memory
$\overline{WR}$/$\overline{WRL}$	P5.2	Write/write low
$\overline{RD}$	P5.3	Read
$\overline{BHE}$/$\overline{WRH}$	P5.5	Byte high enable/write high
READY	P5.6	Ready
BUSWIDTH	P5.7	Bus width
AD0-7	P3.0-7	Multiplexed address/data bus
AD8-15	P4.0-7	

15.1 BUS OPERATION

The 8XC196MC has several external operating modes. The standard bus mode uses a 16-bit multiplexed address/data bus. Other bus modes include an 8-bit external bus mode and a mode in which the bus size can be dynamically switched between 8 and 16 bits. In addition, several bus control signal options make an external bus simple to design.

$\overline{EA}$ = 0/1 directs memory accesses for locations 2000H-5FFFH to external/internal memory, respectively.

In the standard mode, external memory is addressed through lines AD0-AD15 which form a 16-bit multiplexed bus. The address/data bus shares pins with ports 3 and 4. Figure 15.1 shows an idealized timing diagram for the external bus signals.

Address Latch Enable (ALE) provides a strobe to transparent latches (74AC373s) to de-multiplex the bus. The address (''ADDRESS OUT'' on BUS) will be on the bus before ALE drops to strobe the latches. The READY and BUSWIDTH signals are discussed in Sections 15.3.

A read is controlled by $\overline{RD}$. When $\overline{RD}$ falls, the bus is floated to receive the data. The data returned from external memory must be on the bus and stable for a specified setup time before the rising edge of $\overline{RD}$, which signals the end of the sampling window. ''DATA IN'' on BUS indicates that the data is available on the bus. The output signal INST is driven high during an external memory read if the read is an instruction fetch.

A write is controlled by $\overline{WR}$. When $\overline{WR}$ falls, the 8XC196MC puts data on the bus. The data is valid on the rising edge of $\overline{WR}$. At this time the data must be latched by the external system.

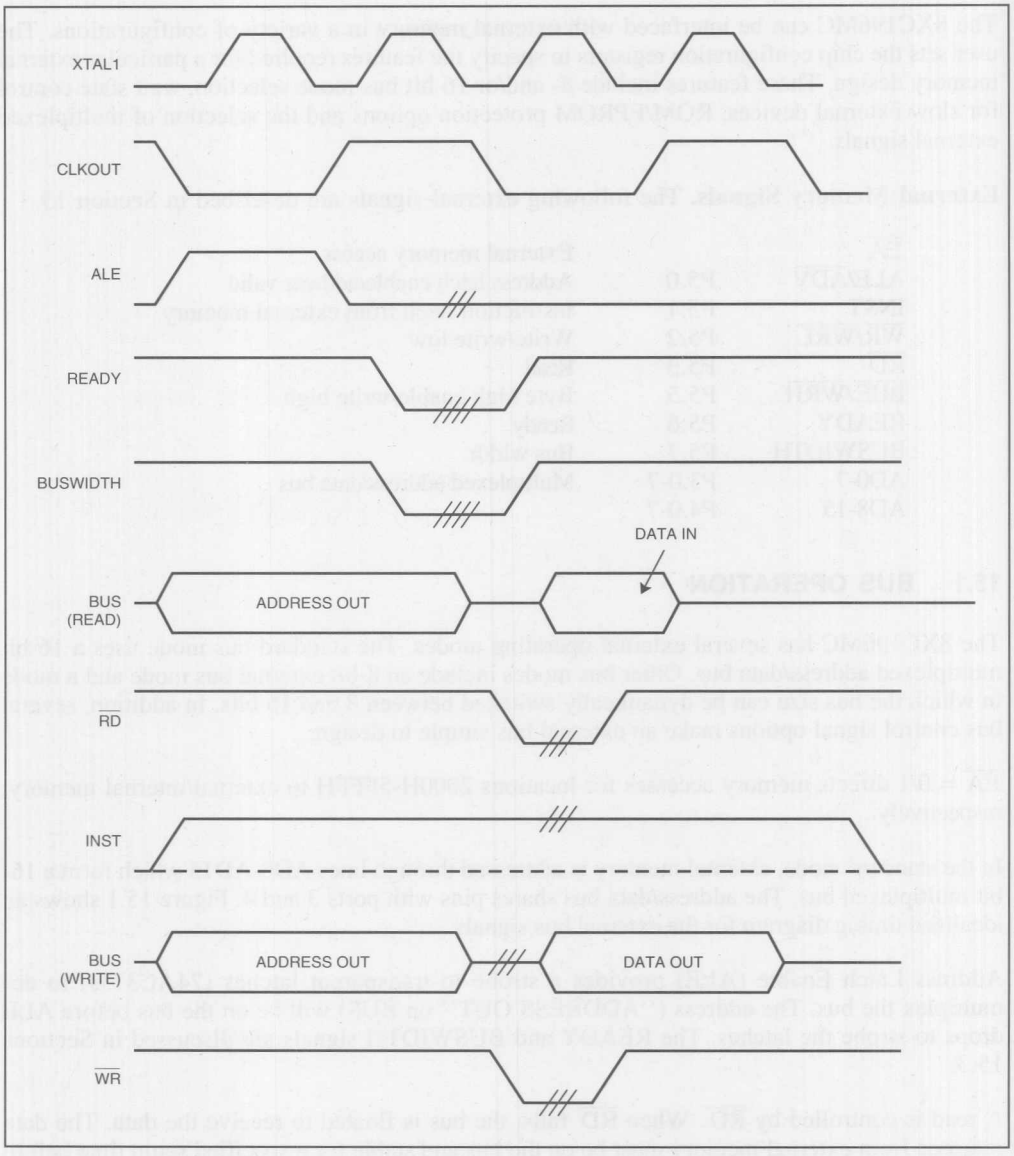


Figure 15.1. Idealized Bus Timings

15.2 CHIP CONFIGURATION REGISTERS

The 8XC196MC has two chip configuration registers (CCRs). The first CCR (CCR0) is the same as the CCR on the 8XC196KB. The second CCR (CCR1) was added to support additional functions. Together they control a variety of memory bus functions.

The chip configuration bytes (CCBs) are the first bytes fetched from memory following a chip reset (Section 14.5). The CCRs are loaded from the CCBs (CCB0 at 2018H and CCB1 at 201AH) in either internal or external memory depending on the $\overline{EA}$ pin. The CCBs are fetched in two consecutive bus cycles to accommodate either an 8- or 16-bit external data bus (Section 15.3.2). The CCRs are only loaded during the reset sequence. Once loaded, the CCRs cannot be changed until the next reset. Figure 15.2 shows the formats of the CCBs (and CCRs).

When implementing 16-bit memory systems, note that after reset but prior to the CCB fetch, the 8XC196MC is configured to perform two byte fetches. In order to prevent contention on 16-bit systems, **the upper byte of the CCB words (2019H and 201BH) must contain 20H**, which is the address out during CCB reads. When the CCBs are read, the bus will be configured as either 8-bit, 16-bit or BUSWIDTH-controlled, as determined by the CCBs.

Most of the CCR bits are for address/data bus control and are discussed in sections to follow. Bits with other functions are PD (for the powerdown mode, Section 13.2), LOC0 and LOC1 (for ROM/EPROM write protection, Section 16.5), and WD (watchdog timer enable, Section 14.5.2).

15.3 BUS CONTROL

This section describes how the bus operation is controlled by the CCR bits and bus signals.

15.3.1 Wait States (Ready Control)

If external memory is too slow to be accessed using the normal bus timings, then wait states can be inserted into the bus cycle to extend the read and write operations. Control of the wait states, called ''ready control,'' is implemented by the external input signal READY and the internal ready control bits IRC0-2.

If the external memory is not ready for access, it pulls the READY signal low and holds it low, causing wait states to be inserted until the memory is ready for access. It then releases READY (see Figure 15.1). If wait states are internally limited, the external memory chip select lines can be ORed together and connected to the READY pin. Otherwise, external control logic is needed. Note that the READY pin (P5.6) must be configured as described in Section 10.2.

	7	6	5	4	3	2	1	0
CCB1	1	1	0	1	WD	BW1	IRC2	0
CCB	LOC1	LOC0	IRC1	IRC0	ALE	$\overline{WR}$	BW0	PD

PD Enables (1) / disables (0) Powerdown Mode (Sec. 13.2)

BW0-1 Bus width control (Sec. 15.3.2, table below)

$\overline{WR}$ 1 = $\overline{WR}$/$\overline{BHE}$. 0 = $\overline{WRL}$/$\overline{WRH}$ (Sec. 15.2, 15.3)

ALE 1 = ALE.0 = $\overline{ADV}$ (Sec. 15.2, 15.3)

IRC0-2 Ready control (Sec. 15.3, table below)

LOC0-1 ROM/EPROM protection (Sec. 16.6, table below)

WD Watchdog Timer Enable. Watchdog enabled the first time it is cleared for WD = 1; watchdog always enabled for WD = 0 (Sec. 14.5.2)

Bus control bits WR and ALE:

WR	ALE	Timing Figure	Description	Example Figure
0	0	15-13	Address valid with write strobe	15-5, 15-7
0	1	15-11	Write strobe mode	15-8
1	0	15-12	Address valid strobe	15-4, 15-6
1	1	15-9	Standard bus control	15-10

BW1	BW0	Bus Width
0	0	Illegal
0	1	16-Bit Only
1	0	8-Bit Only
1	1	Bus Width Pin Controlled

LOC0	LOC1	Bus Width
0	0	Read and Write Protected
1	0	Write Protected Only
0	1	Read Protected Only
1	1	No Protection

IRC2	IRC1	IRC0	Max Wait States
0	0	0	Zero Wait States
0	X	1	Illegal
0	1	X	Illegal
1	0	0	1 Wait State
1	0	1	2 Wait States
1	1	0	3 Wait States
1	1	1	INFINITE

X = Don't care

Figure 15.2. Chip Configuration Bytes

The number of wait states inserted is also influenced by the internal ready control bits IRC0-2, as shown in the logic table of Figure 15.2. You could think of these bits as the output of an internal signal that goes low for the number of wait states dictated by this table. The actual ready signal to the 8XC196MC is the logical OR of this internal signal from IRC0-2 and the external READY signal. Either signal can terminate the insertion of wait states by going high. Thus, to use the READY pin, you must either program the IRC field of the CCBs to limit the wait states, or provide external hardware to count wait states and release READY.

READY (P5.6) is set up as a special function input if IRC0-2 is set to 111B during the CCB fetch. **If port 5 is initialized after reset and IRC0-2 is set to 111B, the user**

must ensure that P5.6 remains as a special function input. If P5.6 is configured as an I/O port pin, the READY input to the chip equals 0. This will cause an infinite number of wait states to be inserted, and the chip will lock up.

After reset (Section 14.5) the bus controller always inserts 3 wait states until CCB1 is read. Wait states are then dictated by IRC0-2 until P5.6 has been configured to operate as READY. Recall that the CCRs are not loaded again from the CCBs until after the next reset. To change the IRC field, you must execute a Reset to make new settings effective. In most applications the CCBs (including IRC0-2) are set at compile time. They are usually not changed during device operation.

15.3.2 Bus Width (BUSWIDTH) and Memory Configurations

The 8XC196MC external bus can operate as a 16-bit multiplexed address/data bus, or a multiplexed 16-bit address/8-bit data bus.

During 16-bit bus cycles, ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, port 3 is multiplexed with address/data, but Port 4 outputs only the upper 8 address bits. The addresses on port 4 are valid throughout the entire bus cycle. Figure 15.3 shows the two bus width options. These widths are determined by the BUSWIDTH pin and CCR bits BW0, BW1.

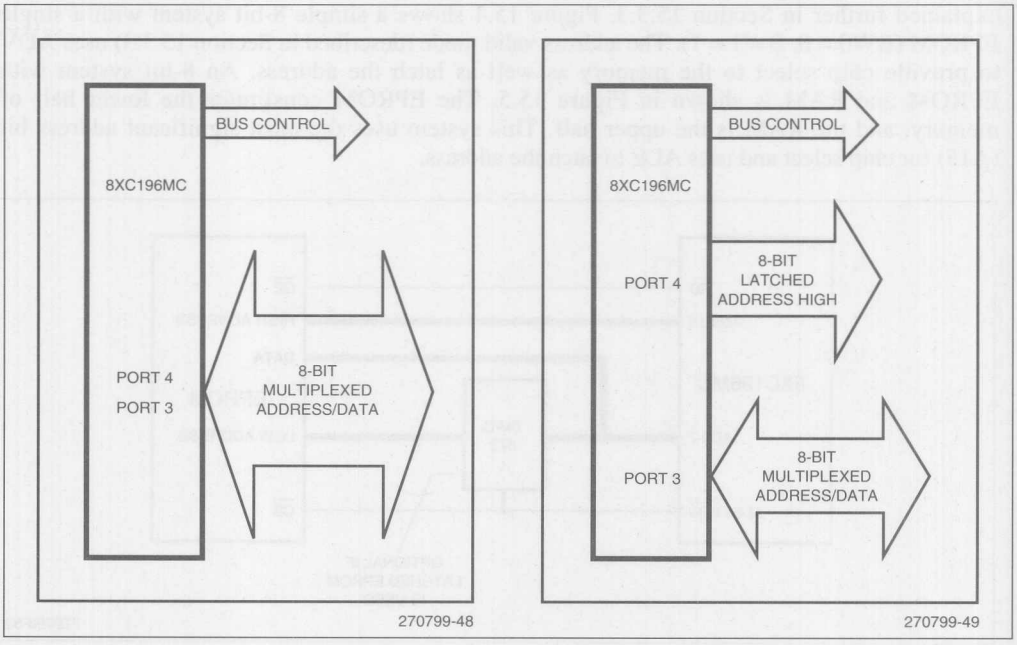


Figure 15.3. Bus Width Options

The external bus width can be changed every bus cycle if BW0 = BW1 = 1 at reset. The bus width is changed on the fly using the BUSWIDTH pin. The bus width is 16 bits for BUSWIDTH = 1 and 8 bits for BUSWIDTH = 0. The BUSWIDTH pin is sampled by the 8XC196MC after the address is on the bus, as shown in Figure 15.1.

Applications using the BUSWIDTH pin are numerous. For example, a system could have code fetched from 16-bit memory and the data stored in 8-bit memory. This saves the cost of using two 8-bit static data RAMs if the capacity of only one is sufficient. This system could be easily implemented by tying the chip select input of the 8-bit memory to the BUSWIDTH pin.

If BW0 = 0 and BW1 = 1, the 8XC196MC is locked into the 8-bit mode and the BUSWIDTH pin is ignored. Some performance degradation is to be expected when executing code from an 8-bit bus. The prefetch queue cannot be kept full under all conditions from an 8-bit bus. Also, word reads and writes to external memory take an extra bus cycle.

If BW0 = 1 and BW0 = 0, the 8XC196MC is locked into the 16-bit bus mode, and BUSWIDTH is ignored.

Memory Configuration Examples

External memory systems for the 8XC196MC can be set up in many ways, as illustrated in Figures 15.4 through 15.8. The signals $\overline{ADV}$, $\overline{WRH}$ and $\overline{WRL}$ are introduced here and explained further in Section 15.3.3. Figure 15.4 shows a simple 8-bit system with a single EPROM (BW0 = 0, BW1 = 1). The address valid mode (described in Section 15.3.3) uses $\overline{ADV}$ to provide chip select to the memory as well as latch the address. An 8-bit system with EPROM and RAM is shown in Figure 15.5. The EPROM constitutes the lower half of memory, and the RAM is the upper half. This system uses the most significant address bit (A15) for chip select and uses ALE to latch the address.

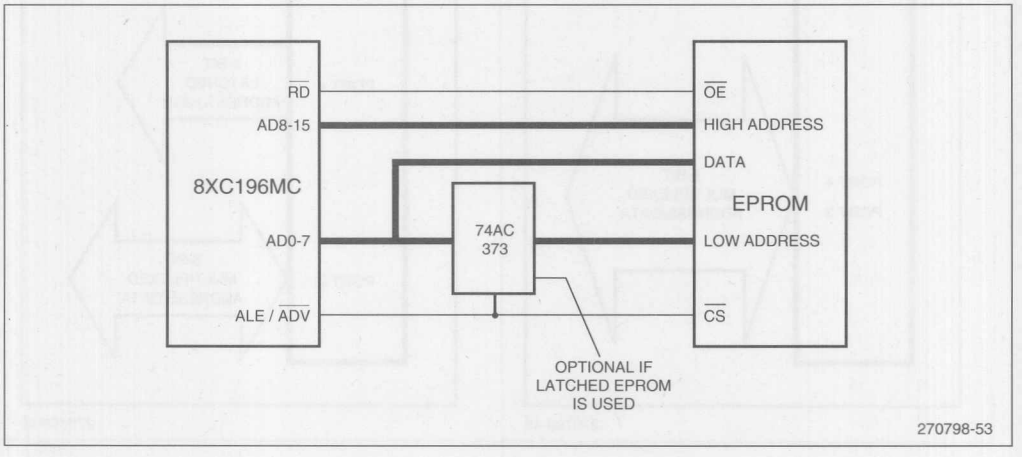


Figure 15.4. 8-Bit System with EPROM

Figure 15.6 shows a 16-bit system with 2 EPROMs (BW0 = 1, BW1 = 0). $\overline{ADV}$ is used to chip select the memory and latch the address. Figure 15.7 shows a system with dynamic bus width (BW0 = BW1 = 1). Code is executed from the two EPROMs and data is stored in the byte-wide RAM. The RAM is the high memory and is selected for A15 = 1, which also selects the 8-bit bus width mode via the BUSWIDTH pin.

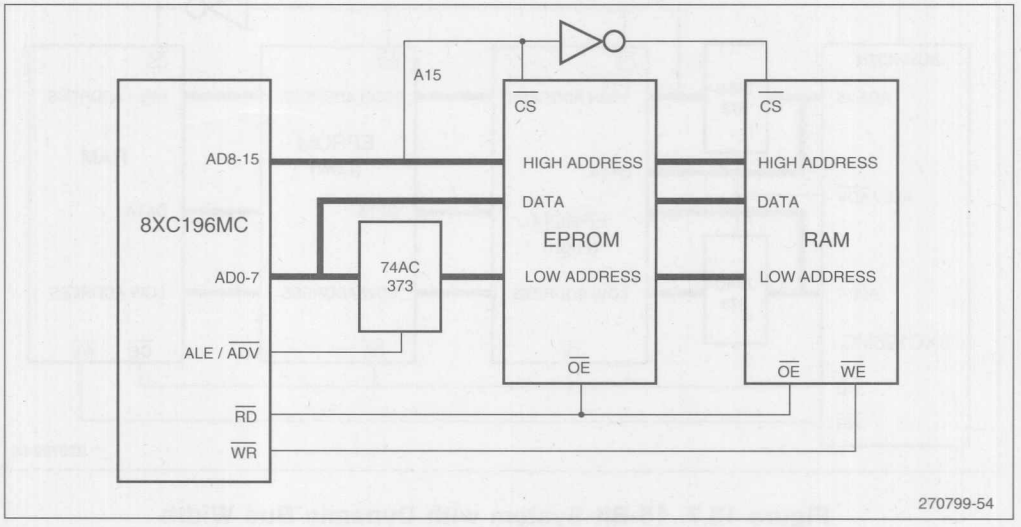


Figure 15.5. 8-Bit System with EPROM and RAM

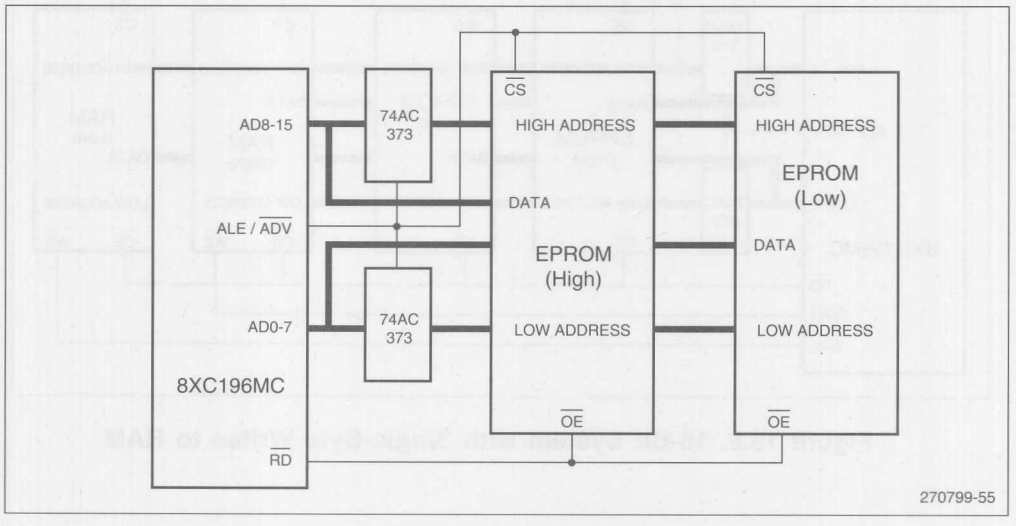


Figure 15.6. 16-Bit System with EPROM

Figure 15.8 shows a 16-bit system with 2 EPROMS and 2 RAMs. A15 is used for chip select. Separate write signals are used for the even bytes ($\overline{WRL}$) and the odd bytes ($\overline{WRH}$). These signals (discussed in Section 15.3.3) allow single-byte writes in the 16-bit system.

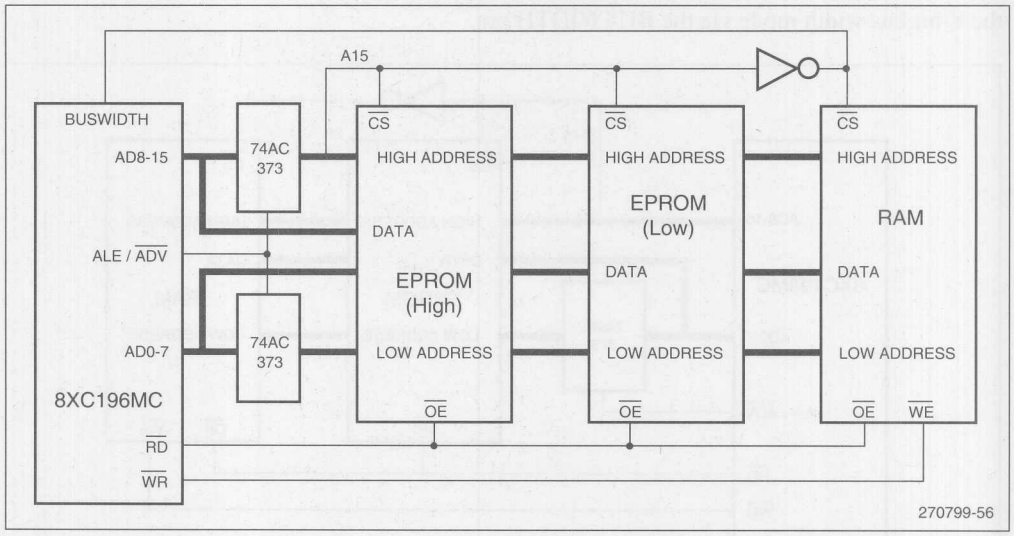


Figure 15.7. 16-Bit System with Dynamic Bus Width

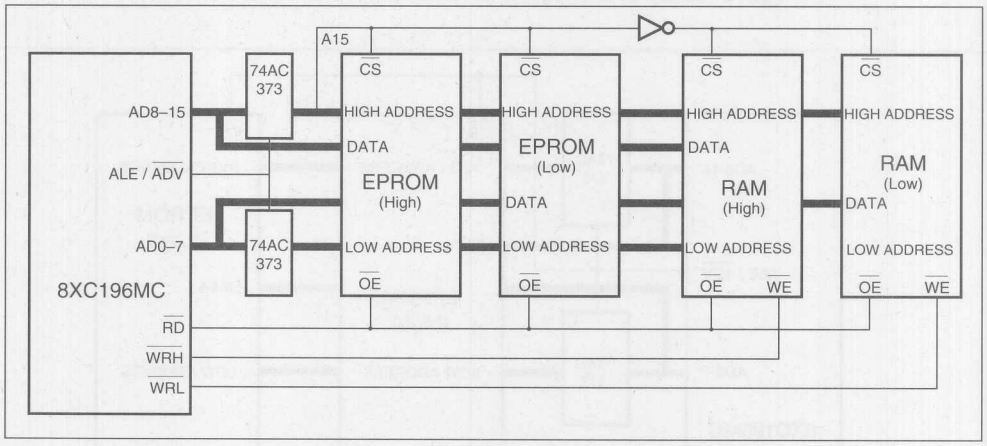


Figure 15.8. 16-Bit System with Single-Byte Writes to RAM

15.3.3 Bus Control Modes

The 8XC196MC can generate several types of control signals designed to reduce external hardware. The $\overline{WR}$ and ALE pins can be configured for multiple functions as shown in Figure 15.2.

Standard Bus Control

If CCB bits $\overline{WR}$ and ALE are both 1, the standard bus control signals ALE, $\overline{WR}$, $\overline{RD}$ and $\overline{BHE}$ (Byte High Enable) are generated as shown in Figure 15.9. ALE rises as the address is driven, and falls to externally latch the address. $\overline{BHE}$ low selects the bank of memory that is connected to the high byte of the data bus. $\overline{RD}/\overline{WR}$ is driven for every read/write. $\overline{BHE}$ and MA0 can be combined to form $\overline{WRL}$ (Write Low) and $\overline{WRH}$ (Write High) for even and odd byte writes. These signals are required for single-byte writes to an external RAM using the 16-bit bus mode. (A similar pair of signals for read is unnecessary. For a single-byte read with the 16-bit bus mode, both bytes are put on the data bus, and the processor discards the unwanted byte.) Figure 15.10 is an example of external circuitry to decode $\overline{WRL}$ and $\overline{WRH}$. The following bus control mode generates $\overline{WRL}$ and $\overline{WRH}$ within the 8XC196MC.

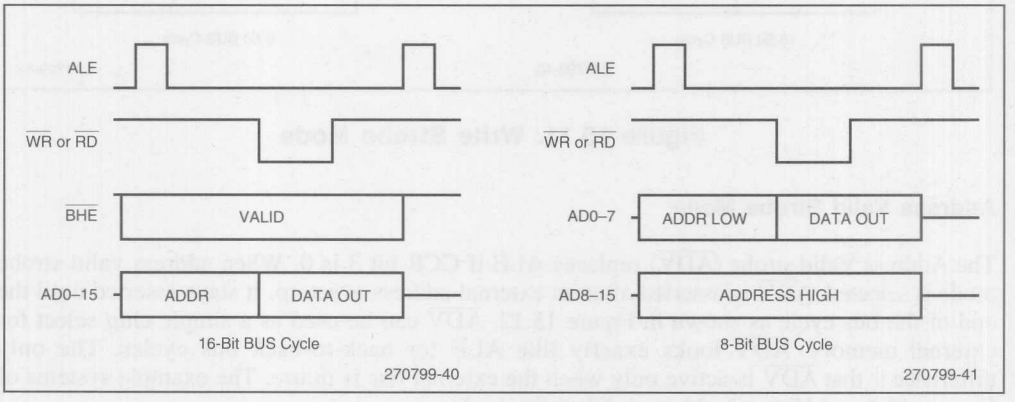


Figure 15.9. Standard Bus Control

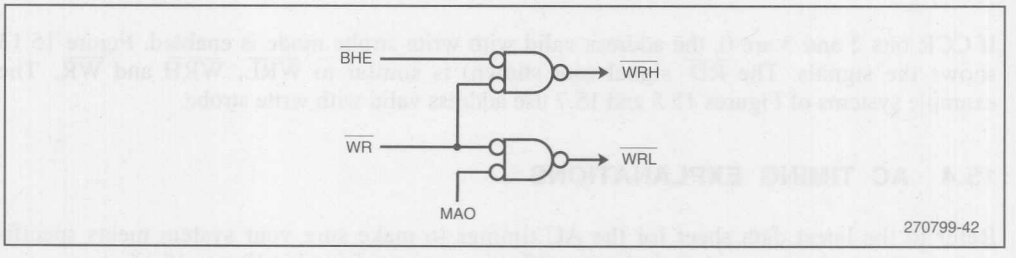


Figure 15.10. Decoding $\overline{WRL}$ and $\overline{WRH}$

Write Strobe Mode

The Write Strobe Mode eliminates the need to externally decode signals for odd and even byte writes. If CCR bit 2 is 0, $\overline{WRL}$ and $\overline{WRH}$ are generated in place of $\overline{WR}$ and $\overline{BHE}$. $\overline{WRL}$ is asserted for all byte writes to an even address and all word writes. $\overline{WRH}$ is asserted for all byte writes to odd addresses and all word writes. In the 8-bit bus mode, $\overline{WRL}$ and $\overline{WRH}$ are asserted for both even and odd addresses. The write strobe mode is shown in Figure 15.11. The 16-bit system of Figure 15.8 uses this mode to write to the odd and even bytes of RAM.

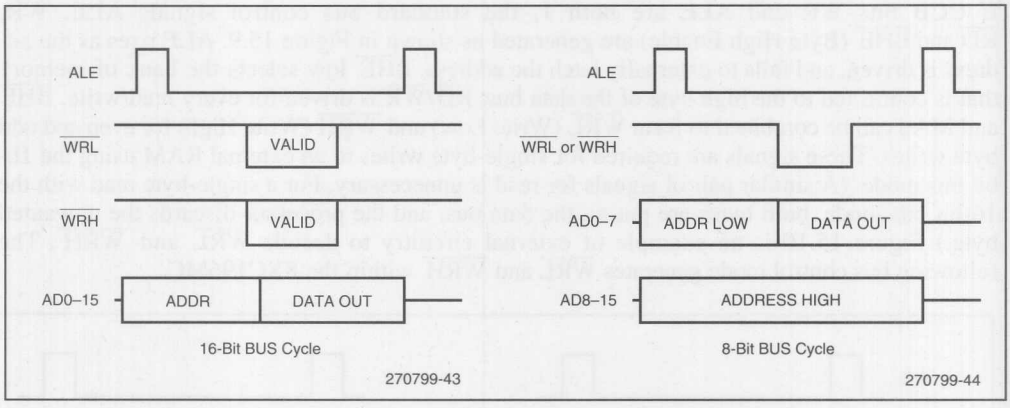


Figure 15.11. Write Strobe Mode

Address Valid Strobe Mode

The Address Valid strobe ($\overline{ADV}$) replaces ALE if CCR bit 3 is 0. When address valid strobe mode is selected, $\overline{ADV}$ is asserted after an external address is set up. It stays asserted until the end of the bus cycle as shown in Figure 15.12. $\overline{ADV}$ can be used as a simple chip select for external memory. $\overline{ADV}$ looks exactly like ALE for back-to-back bus cycles. The only difference is that $\overline{ADV}$ is active only when the external bus is in use. The example systems of Figures 15.4 and 15.6 use address valid strobe mode.

Address Valid with Write Strobe

If CCR bits 2 and 3 are 0, the address valid with write strobe mode is enabled. Figure 15.13 shows the signals. The $\overline{RD}$ signal (not shown) is similar to $\overline{WRL}$, $\overline{WRH}$ and $\overline{WR}$. The example systems of Figures 15.5 and 15.7 use address valid with write strobe.

15.4 AC TIMING EXPLANATIONS

Refer to the latest data sheet for the AC timings to make sure your system meets specifications. The major system bus timing specifications are explained in Figure 15.14.

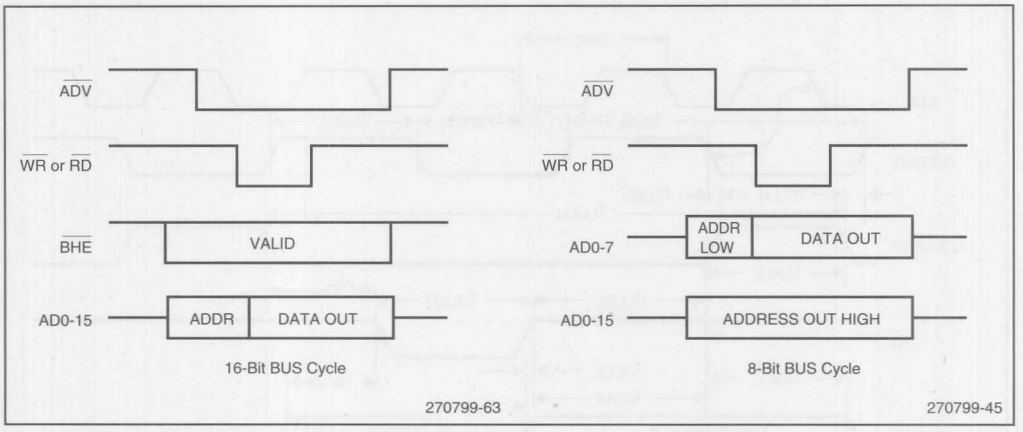


Figure 15.12. Address Valid Strobe Mode

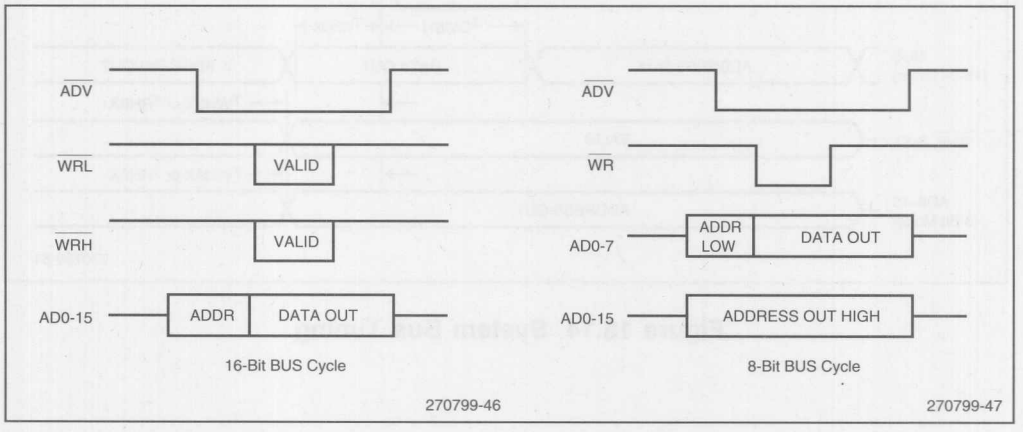


Figure 15.13. Address Valid with Write Strobe Mode

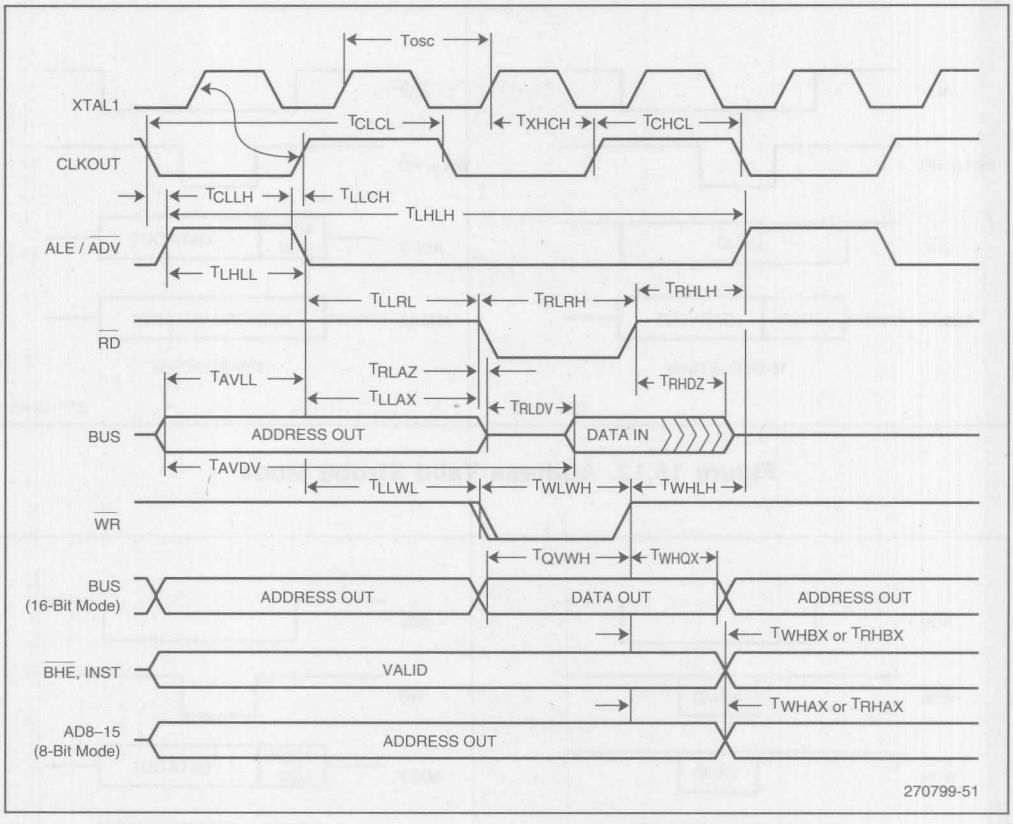


Figure 15.14. System Bus Timing

AC TIMING EXPLANATIONS

TIMINGS THE MEMORY SYSTEM MUST PROVIDE:	
TAVYV	Address Valid to Ready Setup: Maximum time the memory system has to assert READY after the address is output by the 80C196MC to guarantee at least one wait state will occur.
TLLYV	ALE Low to READY Setup: Maximum time the memory system has to assert READY after ALE falls to guarantee at least one wait state will occur.
TCLYX	READY Hold after CLKOUT Low: Minimum hold time is always 0 ns. If maximum spec is exceeded, additional wait states will occur.
TLLYX	READY Hold after ALE Low: Minimum time the level of the READY pin must be valid after ALE falls. If the maximum value is exceeded, additional wait states will occur.
TAVGV	ADDRESS Valid to BUSWIDTH Valid: Maximum time after address is valid until BUSWIDTH must be valid. If exceeded, it is not guaranteed the 80C196MC will respond with an 8- or 16-bit bus cycle.
TLLGV	ALE Low to BUSWIDTH Valid: Maximum time after ALE/ADV falls until BUSWIDTH must be valid. If exceeded, it is not guaranteed the 80C196MC will respond with an 8- or 16-bit bus cycle.
TCLGX	BUSWIDTH Hold after CLKOUT Low: Minimum time BUSWIDTH must be held valid after CLKOUT falls. Always 0 ns on the 80C196MC.
TAVDV	Address Valid to Input Data Valid: Maximum time the memory device has to output valid data after the 80C196MC outputs valid address.
TRLDV	$\overline{RD}$ Low to Input Data Valid: Maximum time the memory system has to output valid data after the 80C196MC asserts $\overline{RD}$.
TCLDV	CLKOUT Low to Input Data Valid: Maximum time the memory system has to output valid data after CLKOUT falls.
TRHDZ	$\overline{RD}$ High to Input Data Float: Time after $\overline{RD}$ is inactive until the memory system must float the bus. If this timing is not met, bus contention will occur.
TRHDX	Data Hold after $\overline{RD}$ High: Time after $\overline{RD}$ is inactive that the memory system must hold data on the bus. Always 0 ns on the 80C196MC.
TIMINGS THE 80C196MC WILL PROVIDE:	
FXTAL	Frequency on XTAL: Frequency of the signal input on the XTAL1 pin input. The internal bus speed of the 80C196MC device is $1/2F_{XTAL}$.
TOSC	1/FXTAL: All AC Timings are referenced to T_{OSC}.
TXHCH	XTAL1 High to CLKOUT High or Low:
TCLCL	CLKOUT Cycle Time: Normally 2 T_{OSC}.
TCHCL	CLKOUT High Period: Needed in systems which use CLKOUT as clock for external devices.
TCLLH	CLKOUT Falling to ALE/$\overline{ADV}$ Rising: A help in deriving other timings.
TLLCH	ALE/$\overline{ADV}$ Falling to CLKOUT Rising: A help in deriving other timings.
TLHLH	ALE Cycle Time: Time between ALE pulses.
TLHLL	ALE/$\overline{ADV}$ High Period. External latch design need this spec.
TAVLL	ADDRESS Setup to ALE/$\overline{ADV}$ Low: Length of time ADDRESS is valid before ALE/$\overline{ADV}$ falls. External latch design need this spec.
TLLAX	ADDRESS Hold after ALE/$\overline{ADV}$ Low: Length of time ADDRESS is valid after ALE/$\overline{ADV}$ falls. External latch design need this spec.
TLLRL	ALE/$\overline{ADV}$ Low to $\overline{RD}$ Low: Length of time after ALE/$\overline{ADV}$ falls before $\overline{RD}$ is asserted. Could be needed to insure proper memory decoding takes place before a device is enabled.
TRLCL	$\overline{RD}$ Low to CLKOUT Low: Length of time from $\overline{RD}$ asserted to CLKOUT falling edge.
TRLRH	$\overline{RD}$ Low to $\overline{RD}$ High: $\overline{RD}$ pulse Width.
TRHLH	$\overline{RD}$ High to ALE/$\overline{ADV}$ Asserted: Time between $\overline{RD}$ going inactive and the next ALE/$\overline{ADV}$. Useful in calculating time between inactive and next ADDRESS valid.

AC TIMING EXPLANATIONS (Continued)

TIMINGS THE MEMORY SYSTEM MUST PROVIDE: (Continued)	
TRLAZ	$\overline{RD}$ Low to ADDRESS Float: Used to calculate when the 80C196MC stops driving ADDRESS on the bus.
TLLWL	ALE/$\overline{ADV}$ Low to $\overline{WR}$ Low: Length of time after ALE/$\overline{ADV}$ falls before $\overline{WR}$ is asserted. Could be needed to insure proper memory decoding takes place before a device is enabled.
TCLWL	CLKOUT Low to $\overline{WR}$ Low: Time between CLKOUT going low and $\overline{WR}$ being asserted.
TQVWH	Data Valid to $\overline{WR}$ High: Time between data being valid on the bus and $\overline{WR}$ going inactive. Memory devices must meet this spec.
TCHWH	CLKOUT High to $\overline{WR}$ High: Time between CLKOUT going high and $\overline{WR}$ going inactive.
TWLWH	$\overline{WR}$ Low to $\overline{WR}$ High: $\overline{WR}$ Pulse Width.
TWHQX	Data Hold after $\overline{WR}$ High: Length of time after $\overline{WR}$ rises that the data stays valid on the bus. Memory devices must meet this spec.
TWHLH	$\overline{WR}$ High to ALE/$\overline{ADV}$ High: Time between $\overline{WR}$ going inactive and next ALE/$\overline{ADV}$. Also used to calculate $\overline{WR}$ inactive and next ADDRESS valid.
TWHBX	BHE, INST, Hold after $\overline{WR}$ High: Minimum time these signals will be valid after $\overline{WR}$ inactive.

Using ROM and EPROM Parts

16

CHAPTER 16
USING ROM AND EPROM PARTS

The 87C196MC contains 16 Kbytes of ultraviolet Erasable Programmable Read Only Memory (EPROM) at locations 2000H-5FFFH. The 87C196MC supports three programming modes.

- The auto programming mode enables the 8XC196MC to program itself from an external EPROM without an EPROM programmer.
- The slave mode provides a standard interface for programming by an EPROM programmer.
- The run-time mode allows individual EPROM locations to be programmed at run-time under complete software control. Unlike the other modes, run-time programming is accomplished without entering the general EPROM programming mode.

16.1 POWER-UP AND POWER-DOWN

To avoid damaging the device during programming the following rules should be followed:

1) V_{PP} must be within 1 volt of V_{CC} while V_{CC} is less than 4.5V. V_{PP} cannot go above 4.5V until V_{CC} is at least 4.5V.
2) $\overline{EA}$ must be brought up to the programming voltage before V_{PP}.
3) The PMODE pins must be in their desired state before $\overline{RESET}$ rises.
4) All voltages must be within the range stated in the data sheet and the oscillator must be stable before RESET rises.
5) The supplies to V_{CC}, V_{PP}, $\overline{EA}$ and RESET must be well regulated and free of glitches and spikes.

The following is the recommended sequence when powering-up or powering-down the device for programming:

Power-Up

1) $\overline{RESET} = 0V$
2) $V_{CC} = V_{PP} = \overline{EA} = 5V$
3) CLOCK on (if using an external clock instead of internal oscillator)
4) $\overline{PALE} = \overline{PROG} = PBUS = V_{IH}$ (2.4V minimum)
5) PMODE valid
6) $\overline{EA}$ = 12.5V* (nominal voltage – see data sheet for current specifications)

7) V_{PP} = 12.5V** (nominal voltage – see data sheet for current specifications)

8) Wait for supplies to settle and oscillator to stabilize

9) $\overline{RESET}$ = 5V

10) Wait for T_{SHLL} (see data sheet)

11) Begin programming

Power-Down

1) $\overline{RESET}$ = 0V

2) V_{PP} = 5V

3) $\overline{EA}$ = 5V

4) $\overline{PALE}$ = $\overline{PROG}$ = PMODE = PBUS = 0V

5) V_{CC} = V_{PP} = $\overline{EA}$ = 0V

6) Clock off

* The same power supply can be used for $\overline{EA}$ and V_{PP}. However, $\overline{EA}$ must be powered-up before V_{PP}. Also, $\overline{EA}$ should be protected from voltage spikes to prevent device damage.

** Exceeding the maximum limit on V_{PP} for any amount of time could damage the device permanently. The V_{PP} source must be well regulated and free of glitches and spikes.

16.2 PROGRAMMING THE 87C196MC

Applying 12.5V to $\overline{EA}$ when $\overline{RESET}$ is low places the 87C196MC into the EPROM programming mode, which supports programming and verification of 87C196MC EPROMs. As the device is executing internal test code in this mode, minimum hardware connections must be made, i.e., XTAL1 driven, unused input pins strapped and power and grounds applied. See Section 14 for the required connections. 87C196MC EPROMs are programmed as specified in the operating conditions in the programming sections of the data sheet. V_{PP} is nominally 12.5V and V_{CC} is nominally 5.0V.

In the programming mode some I/O pins have new functions. These new pin functions determine and support the different programming modes. Figure 16.1 shows how the pins are renamed, and Table 16.1 describes each new pin function. The states of pins P0.4-7 define the hex value of PMODE, which determines the programming mode used. An exception is run-time programming, which can be done at any time during normal execution. Table 16.2 lists the values of PMODE and the corresponding modes.

To guarantee proper operation, the pins of PMODE must be in their desired state before $\overline{RESET}$ rises. The 8XC196MC should not be switched to another mode without executing a new reset sequence.

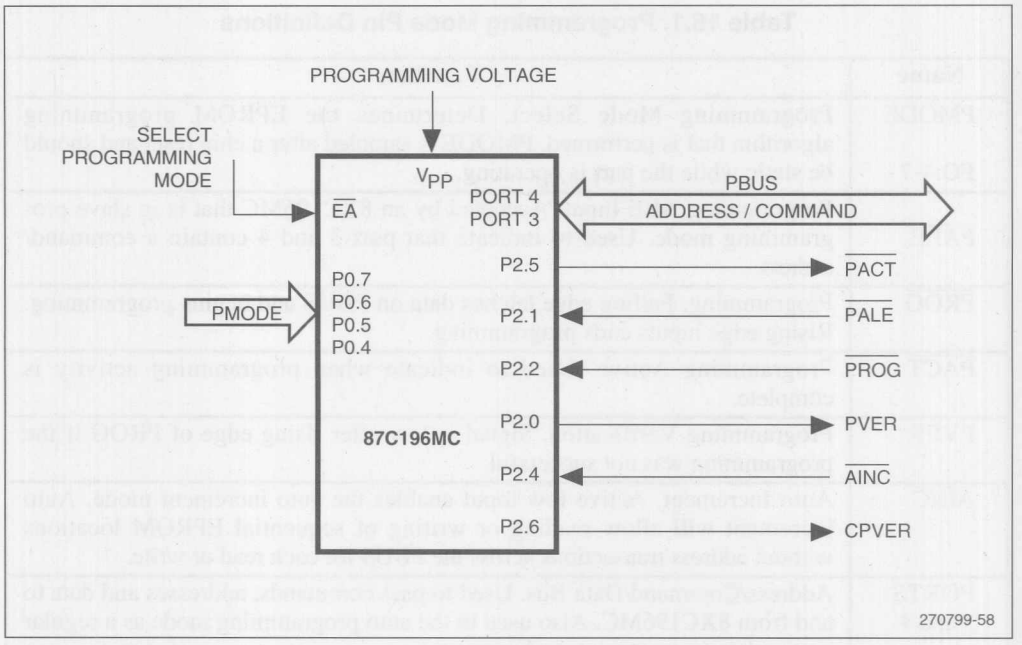


Figure 16.1. Programming Mode Pin Functions

When $\overline{EA}$ is at 12.5V the programming mode is selected. The chip reset sequence loads the lower byte of the chip configuration register (CCR) from the programming configuration byte (PCCB) rather than from the lower chip configuration byte (CCB). PCCB is an EPROM location that is accessible only in programming mode. PCCB has implications in some of the programming modes, and for the memory protection options discussed in Section 16.7. Therefore, the PCCB must correspond to the memory system in the **programming** setup, which is not necessarily the memory system of the application.

16.3 AUTO PROGRAMMING MODE

The auto programming mode allows an 87C196MC EPROM to be programmed without an EPROM programmer. In this mode, the 87C196MC programs itself with the data at external locations 4000H-7FFFH. Figure 16.2 shows how to use a 16K × 8 EPROM to program an 87C196MC in the auto programming mode.

Auto programming is supported for 8-bit bus systems only. If using the default PCCB value of 0FFH, the BUSWIDTH pin must be held low to force an 8-bit bus width.

Table 16.1. Programming Mode Pin Definitions

Name	Function
PMODE PO.4-7	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
$\overline{\text{PALE}}$	Programming ALE Input. Accepted by an 8XC196MC that is in slave programming mode. Used to indicate that port 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming. Falling edge latches data on PBUS and begins programming. Rising edge inputs ends programming.
$\overline{\text{PACT}}$	Programming Active. Used to indicate when programming activity is complete.
PVER	Programming Verification. Signal is low after rising edge of PROG if the programming was not successful.
$\overline{\text{AINC}}$	Auto Increment. Active low input enables the auto increment mode. Auto increment will allow reading or writing of sequential EPROM locations without address transactions across the PBUS for each read or write.
PORTS 3 and 4	Address/Command/Data Bus. Used to pass commands, addresses and data to and from 8XC196MC. Also used in the auto programming mode as a regular system bus to access external memory.
CPVER	Cumulative Program Verification. Pin is high if all locations since entering a programming mode have programmed correctly.

Table 16.2. Programming Mode PMODE Values

PMODE	Programming Mode
0	Reserved
1-4	Reserved
5	Slave Programming
6	ROM Dump
7-8	Reserved
9	UPROM Programming
0AH-0BH	Reserved
0CH	Auto Programming
0DH	PCCB Programming
0EH-0FH	Reserved

When the auto programming mode is entered, the 8XC196MC asserts $\overline{\text{PACT}}$ and configures an internal 16-bit programming pulse register (PPR) to set the programming pulse width (PPW).

The PPR is loaded from external memory location 4014H (low byte) and 4015H (high byte). In order for the EPROM to program properly, the PPW must be set according to the data sheet specifications. The PPR value is calculated using the following formula:

$$\text{PPR (word value)} = \frac{((\text{PPW}) \times (\text{FOSC})) - 144}{144} + 32768$$

NOTE

32768 is added because the most significant bit of the PPW word must always be set to 1.

The 8XC196MC then reads a word from external memory, and the modified quick-pulse programming algorithm (Section 16.8.1) programs the corresponding EPROM location. Since the erased state of a byte is 0FFH, the auto programming mode skips locations containing 0FFH. PVER is driven low anytime a programming error occurs. Each location to be programmed in the EPROM array is pulsed 25 times before moving to the next location. When all 16 Kbytes have been programmed, $\overline{\text{PACT}}$ is driven high.

16.4 SLAVE PROGRAMMING MODE

An 87C196MC can be programmed by a master programmer using the slave programming mode. In this mode, the 87C196MC programs like an EPROM device. The 87C196MC responds to two different commands while in this mode: data program and word dump. These commands, along with the transfer of the appropriate addresses and data, are selected using ports 3 and 4 (PBUS) and some of the port 2 pins for handshaking (Figure 16.1). The LSB of port 3 selects data program command (P3.0 = 1) or word dump command (P3.0 = 0). The upper 15 bits contain the address to be programmed or dumped. Thus, the 16 bits together constitute a combined address-command. The address ranges from 2000H to 5FFFH and refers to **words** in internal EPROM space.

16.4.1 Data Program Command

The data program command is selected by setting the LSB of the address (P3.0). For example, setting P3, P4 to 3501H programs a word of data at location 3500H.

Figure 16.3 illustrates an example of data program command signals. Asserting $\overline{\text{PALE}}$ latches the command-address to be programmed on ports 3 and 4. Asserting $\overline{\text{PROG}}$ latches the data to be programmed on ports 3 and 4. The width of $\overline{\text{PROG}}$ determines the length of the programming pulse. After the rising edge of $\overline{\text{PROG}}$, the 87C196MC asserts PVER if the location was programmed correctly. This verification information is useful to programmers who cannot use the word dump command. $\overline{\text{PROG}}$ can be pulsed to repeat programming. $\overline{\text{CPVER}}$ is a cumulative program verify that remains low if a location failed verification. The $\overline{\text{AINC}}$ pin can be toggled to auto-increment the address to the next word location. If $\overline{\text{AINC}}$ is not toggled, a new data program command must be sent.

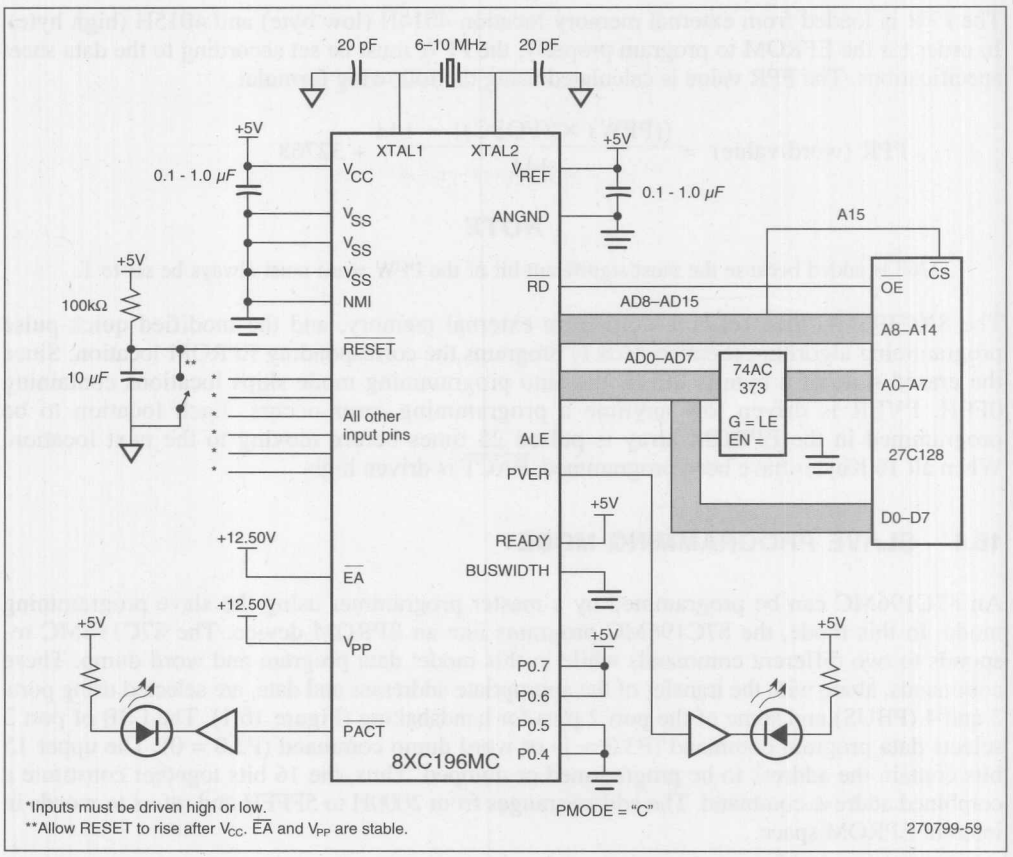


Figure 16.2. Auto Programming Mode

16.4.2 Word Dump Command

In the slave programming mode, a 0 on port pin 3.0 selects the word dump command. The 8XC196MC reads the word at the location specified and places it on Ports 3 and 4. For example, if the command 2100H is sent to the 8XC196MC slave, the 8XC196MC puts the word at internal address 2100H on ports 3 and 4.

The sequence is shown in Figure 16.4. The 8XC196MC drives the bus when $\overline{\text{PROG}}$ goes low. In the word dump mode, if $\overline{\text{AINC}}$ remains active, then asserting $\overline{\text{PROG}}$ pin automatically increments the address to the next word.

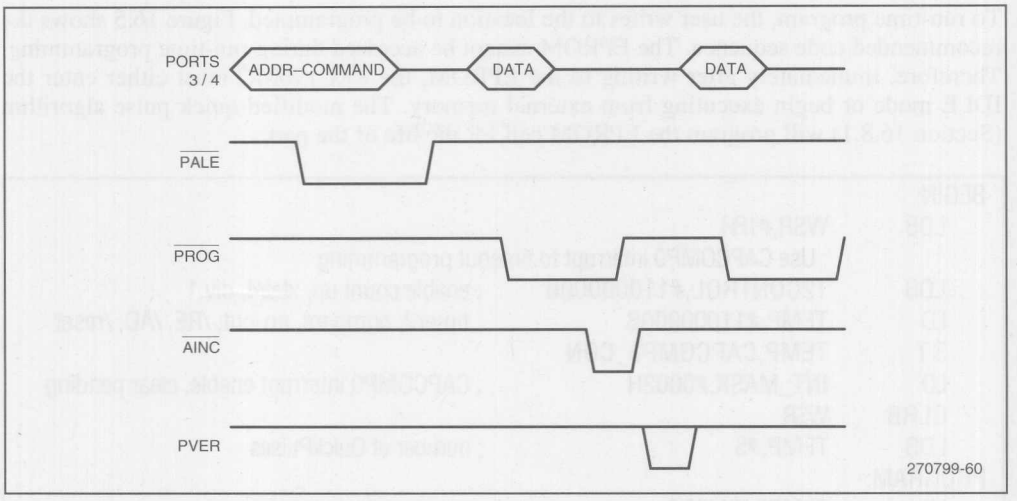


Figure 16.3. Data Program Command

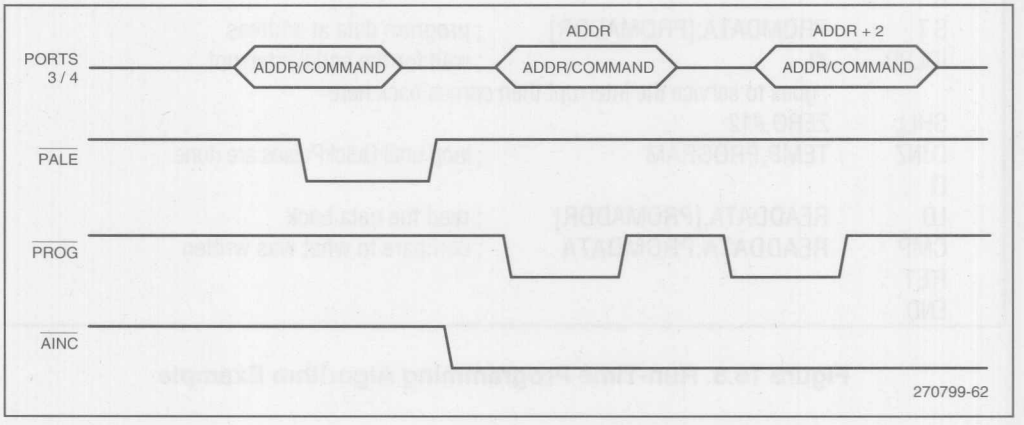


Figure 16.4. Word Dump Command

16.5 RUN-TIME PROGRAMMING

In run-time programming, the user can program an EPROM location during the normal execution of code. The only additional requirement for run-time programming is to have the programming voltage applied to V_{PP} (12.5V). Run-time programming is done with $\overline{EA}$ at a TTL high ($2.4V < EA < V_{CC}$).

To run-time program, the user writes to the location to be programmed. Figure 16.5 shows the recommended code sequence. The EPROM cannot be accessed during run-time programming. Therefore, immediately after writing to the EPROM, the 8XC196MC must either enter the IDLE mode or begin executing from external memory. The modified quick pulse algorithm (Section 16.8.1) will program the EPROM cell for the life of the part.

```
BEGIN:
    LDB     WSR,#1FH
            ; Use CAPCOMP0 interrupt to timeout programming
    LDB     T2CONTROL,#11000000B    ; enable count up, xtal/4, div.1
    LD      TEMP,#11000000B         ; timer2, compare, no-out, /RE, /AD, /reset
    ST      TEMP,CAPCOMP3_CON
    LD      INT_MASK,#0002H         ; CAPCOMP0 interrupt enable, clear pending
    CLRB    WSR
    LDB     TEMP,#5                 ; number of QuickPulses
PROGRAM:
    LD      TEMP2,TIMER2            ; cause a interrupt here
    ADD     TEMP2,#400
    ST      TEMP2,CAPCOMP0_TIME
    EI
    ST      PROMDATA,[PROMADDR]     ; program data at address
    IDLPD   #1                      ; wait for the serial interrupt
            ; goes to service the interrupt then comes back here
    SHLL    ZERO,#12
    DJNZ    TEMP,PROGRAM            ; loop until QuickPulses are done
    DI
    LD      READDATA,[PROMADDR]     ; read the data back
    CMP     READDATA,PROMDATA       ; compare to what was written
    RET
    END
```

Figure 16.5. Run-Time Programming Algorithm Example

16.6 ROM/EPROM MEMORY PROTECTION OPTIONS

Write protection is available for EPROM parts and read protection is provided for both ROM and EPROM parts. This protection is controlled by the LOC0 and LOC1 bits of CCB (Section 15.2) and PCCB and also by bits of the UPROM special function register (USFR) described in Section 16.7.

Write protection is enabled by setting LOC0 (bit 6) in either CCB or PCCB to 0. Table 16.3 summarizes the different write protection options. With write protection enabled, the bus controller cycles through the write sequence but does not actually drive data to the EPROM or

enable V_{PP}. This protects the entire EPROM (2000H-5FFFH) from inadvertent or unauthorized programming. With write protection enabled, the PTS cannot read the internal EPROM.

Table 16.3. Write Protection Options

CCB/PCCB		Protection
LOC1	LOC0	
1	1	Unprotected EPROM. Writes to internal memory always allowed.
1	0	Run-Time Programming Allowed. No programming modes allowed.
0	1	Run-Time programming not allowed. Programming modes allowed after key verification (if needed)
0	0	All programming unconditionally disabled.

Read protection is enabled by clearing the LOC1 (bit 7) of the CCB. With read protection enabled, the bus controller will only perform a data read from address ranges 2020H-202FH and 2050H-5FFFH. Since the Slave PC can be as many as 4 bytes ahead of the CPU program counter, an instruction after address 5FFAH may not access protected memory. The interrupt vectors and CCBs are not read protected because interrupts can occur even when executing out of external memory.

Also, two UPROM (Unerasable PROM) bits are implemented on the 8XC196MC for additional memory protection. (The UPROMS are described in Section 16.7.) Clearing the DEI (disable external instruction fetch) bit prevents the bus controller from executing external instruction fetches. If an attempt is made to load the Slave PC with an external address, the 8XC196MC will reset itself. The automatic reset also gives extra protection against runaway code. If this feature is enabled, code cannot be executed from the last four bytes of internal memory due to the prefetch queue in the bus controller.

Clearing the DED (disable external data fetch) bit prevents the bus controller from executing external data reads and writes. If a data access is requested from the bus controller, the 8XC196MC will reset itself.

16.6.1 Authorized Access of Protected Memory

A ''security key'' mechanism has been implemented for authorized access of protected internal memory to test internal ROM/EPROM. This allows users to verify the EPROM array and still keep their code protected.

The security key is a 128-bit number located in internal memory at locations 2020H-202FH. The user programs his own security key into these locations. Table 16.4 shows the conditions for a security key verification before entering each of the programming modes.

Table 16.4. Security Key Verification

Programming Mode	Key Verification
Slave Programming	If Protected*
ROM Dump	Always
AUTO Programming	If Protected*

*If read or write protected in CCR

If the PCCB is programmed with read and/or write protection (bit 6 =0 or bit 7 = 0), none of the programming modes can be entered. So, to protect the part from unauthorized access, program the PCCB as the last step in preparing the 8XC196MC.

If the CCB has read and/or write protection enabled (LOC0 = 0 or LOC1 = 0), the security key is write protected to keep unauthorized users from overwriting the key with a known security key.

To use the auto programming mode, the security key must be verified if the CCB has read and/or write protection enabled. The security key must reside in external memory locations 4020H-402FH, and if the two keys do not match, the 8XC196MC enters an endless internal loop.

In the slave programming mode, a security key verification is also performed if the CCB has read and/or write protection enabled. The key is verified somewhat differently in the slave programming mode. The user must ''program'' the correct key into the array at locations 2020H-202FH using the data program command described in Section 16.4.1. The locations are not actually programmed, but the data is compared to the internal security key. The key must be entered sequentially, and if any of the locations is ''programmed'' incorrectly, the 8XC196MC enters an endless internal loop.

16.6.2 ROM Dump Mode

The ROM dump mode is an easy way to verify the contents of the EPROM or ROM array. The ROM dump writes out the entire EPROM or ROM to locations 4000H-7FFFH in external memory. If the DED (disable external data fetches) bit has been programmed in the USFR, this mode is disabled entirely (see Section 16.7).

The ROM dump mode is selected with PMODE = 06H and always begins with a security key verification. The user puts into external locations 4020H-402FH the same security key that he has programmed in internal locations 2020H-202FH. Before doing a ROM dump, the 8XC196MC compares the two security keys. If they match, the 8XC196MC dumps the array to external memory. If they do not match, the 8XC196MC enters an endless loop of internal execution which can be exited only by a chip reset.

16.7 UPROMs

Unerasable PROM (UPROM) bits are implemented on the 8XC196MC for additional security features. The UPROM special function register (USFR, location 1FF6H) is shown in Figure 16.6, which specifies which bits are available for the ROM and EPROM versions. The UPROM bits act as fuses, i.e., they can be programmed but not erased. The USFR can be read from location 1FF6H but is written as described in the next section on programming UPROMs.

USFR: Byte, Location 1FF6H (for read only)

7	6	5	4	3	2	1	0
RSV	RSV	RSV	DEI	DED	RSV	RSV	RSV

Note: Do not write to location 1FF6H. Bits DED and DEI are written as specified in Section 16.7 .1.

Device	DEI	DED
87C196MC	UPROM Bit	UPROM Bit
83C196MC	N/A	N/A

Figure 16.6. UPROM Special Function Register (USFR)

Because the UPROM bits cannot be erased, the bits cannot be tested by Intel prior to shipment. Intel does, however, test the features enabled by the UPROM bits. Therefore, the only undetectable defects are those (highly unlikely) defects within the UPROM cells themselves. When a UPROM bit is programmed, its contents can be verified.

WARNING

Once a UPROM bit is programmed it cannot be erased. Programming these bits makes it impossible for Intel (or anyone else) to perform dynamic failure analysis. Therefore, devices with programmed UPROM bits cannot be returned to Intel for failure analysis.

16.7.1 Programming UPROMs

There are two UPROM bits: disable external data fetches (DED) and disable external instruction fetch (DEI). The UPROM bits in the USFR can be programmed using the slave programming mode.

To program the USFR in the slave programming mode, the USFR bits are programmed at separate locations with the following data:

Bit	Address	Data
DED	0758H	0004H
DEI	0718H	0008H

16.8 Algorithms

16.8.1 Modified Quick Pulse Algorithm

The modified quick pulse algorithm must be used to guarantee programming over the life of the EPROM in the run-time and slave programming modes.

The modified quick-pulse algorithm calls for each EPROM location to receive 25 separate programming pulses. Refer to the data sheet to determine the period of each programming pulse. The programming is verified after the 25th pulse. If the data in the location is verified, the next location is programmed. If the data fails verification, the location fails the programming sequence. Once all locations are programmed and verified, the entire EPROM is again verified.

16.8.2 Signature Word

The 8XC196MC contains a signature word. The word can be accessed in the slave mode by executing a word dump command at memory location 70H. The programming voltages are determined by reading locations 72H and 73H in slave programming mode. The voltages are calculated by using the following equation:

$$\text{Voltage} = (20 \times \text{Test ROM Data})/256$$

The values for the signature word and voltage levels are shown in Table 16.5.

Table 16.5. Signature Word and Voltage Levels

Description	Location	Value
Signature Word	70H	8795H
Program V_{CC}	72H	040H 5.0V
Program V_{PP}	73H	0A0H 12.50V

Appendix A
Pin Descriptions

APPENDIX A
PIN DESCRIPTIONS

SYMBOL	DESCRIPTION
ACH0-ACH12 (P0.0-P0.7, P1.0-P1.4)	Analog inputs to the on-chip A/D converter. ACH0-7 share the input pins with P0.0-7 and ACH8-12. Share pins with P1.0-4. If the A/D is not used, the port pins can be used as standard input ports.
AINC (P2.4)	Auto Increment: Input to enable the automatic increment of the address when in programming mode.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as Vss.
ALE/$\overline{\text{ADV}}$ (P5.0)	Address Latch Enable or Address Valid output, as selected by CCR. Both options allow a latch to demultiplex the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive (high) at the end of the bus cycle. When the pin is ALE, the address can be latched on the falling edge. ALE/$\overline{\text{ADV}}$ is active only during external memory accesses. Can be used as standard I/O when not used as ALE/ADV.
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$ (P5.5)	Byte High Enable or Write High output, as selected by the CCR. $\overline{\text{BHE}}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. If the $\overline{\text{WRH}}$ function is selected, the pin will go low when the bus cycle is writing to an odd memory location. $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is only valid during 16-bit external memory cycles. Can be used as standard I/O when not used as a bus control signal.
BUSWIDTH (P5.7)	Input for bus width selection. If CCR bits 1 and 2 = 1, this pin dynamically controls the bus width of the bus cycle in progress. If BUSWIDTH is low, an 8-bit cycle occurs. If it is high, a 16-bit cycle occurs. This pin can be used as standard I/O when not used as BUSWIDTH.
CAPCOMP0-CAPCOMP3 (P2.0-P2.3)	The EPA Capture/Compare pins. These pins share P2.0-P2.3. If not used for EPA, they can be configured as standard I/O pins.
CLKOUT	Output of the internal clock generator. The frequency is 1/2 of the oscillator frequency. If has a 50% duty cycle.

SYMBOL	DESCRIPTION
COMPARE0-COMPARE3 (P2.4-P2.7)	The EPA Compare pins. These pins share P2.4-P2.7. If not used for EPA, they can be configured as standard I/O pins.
CPVER (P2.6)	Cumulative Program Verification: Indicates when all EPROM locations program correctly.
$\overline{EA}$	External Access enable pin. $\overline{EA}$ = 0 causes all memory accesses to be external to the chip. $\overline{EA}$ = 1 causes memory accesses from locations 2000H to 5FFFH to be to the on-chip OTPROM/ROM. $\overline{EA}$ = 12.5 V causes execution to begin in the programming mode. $\overline{EA}$ is latched at reset.
EXTINT	A programmable input on this pin causes a maskable interrupt vector through memory location 203CH. The input may be selected to be a positive/negative edge or a high/low level.
INST (P5.1)	INST is high during the instruction fetch from the external memory and throughout the bus cycle. It is low otherwise. This pin can be configured as standard I/O if not used as INST.
NMI	A positive transition on this pin causes a non-maskable interrupt which vectors to memory location 203EH. If not used, it should be tied to Vss. May be used by Intel evaluation boards.
$\overline{PACT}$ (P2.5)	Output that indicates when the device is currently programming itself. Not active during slave programming
$\overline{PALE}$ (P2.1)	Programming Address Latch Enable: Input to latch the address during programming modes.
PBUS (Port3, Port4)	Programming Bus: Command, address, and data bus during programming. When the address is on the bus, P3.0 is the command bit.
PMODE.0-3 (P0.4-7)	Programming mode select inputs.
PORT0	8-bit impedance input-only port. Also used as A/D converter inputs (ACH8-12). Port0 pins should not be left floating. These pins are also used to select programming modes in the OTPROM devices.

SYMBOL	DESCRIPTION
PORT1	5-bit high impedance input-only port. P1.0-P1.4 are also used as A/D converter inputs(ACH0-7). In addition, P1.2 and P1.3 can be used as Timer1 clock input and direction select respectively (T1CLK and T1DIR).
PORT2	8-bit bidirectional I/O port. All of Port2 pins are shared with the EPA I/O pins (CAPCOMP0-3 and COMPARE0-3).
PORT3, PORT4	8-bit bidirectional I/O ports with open drain outputs. These pins are shared with the mutliplexed address/data bus which uses strong internal pullups.
PORT5	8-bit bidirectional I/O port. 7 of the pins are shared bus control signals ($\overline{ALE}$, INST, $\overline{WR}$, $\overline{RD}$, $\overline{BHE}$, READY, BUSWIDTH). Can be used as standard I/O
PORT6	8-bit output-only port. P6.6 and P6.7 output PWM and the rest are used as the Wave Generator outputs. Can be used as standard output ports.
$\overline{PROG}$ (P2.2)	Programming Mode enable input.
PWM0, PWM1 (P6.6, P6.7)	Programmable duty cycle; programmable frequency Pulse Width Modulator pins. The duty cycle has a resolution of 256 steps, and the frequency can vary from 122 Hz to 31 KHz (16 MHz input clock. Pins may be configured as standard output if PWM is not used.
PVER (P2.0)	Program Verification: Goes high level after a byte/word is programmed to indicate a successful operation.
$\overline{RD}$ (P5.3)	Read signal output to external memory. $\overline{RD}$ is low only during external memory reads. Can be used as standard I/O when not used as $\overline{RD}$.
READY (P5.6)	Ready input to lengthen external memory cycles. If READY = 0, the memory controller inserts wait states until the next positive transition of CLKOUT occurs with READY = 1. Can be used as standard I/O when not used as READY.
$\overline{RESET}$	Reset input to the chip. Held low for at least 16 state times to reset the chip. Input high for normal operation. $\overline{RESET}$ has an Ohmic internal pullup resistor.

SYMBOL	DESCRIPTION
T1CLK (P1.2)	Timer 1 Clock input. This pin has two other alternate functions: ACH10 and P1.2
T1DIR (P1.3)	Timer 1 Direction input. This pin has two other alternate functions: ACH11 and P1.3
V_{PP}	The programming voltage is applied to this pin. It is also the timing pin for the return from Power Down dircuit. Connect this pin with a 1 µF capacitor to V_{SS} and a 1 MOhm resistor to V_{CC}. If the Power Down feature is not used, connect the pin to V_{CC}.
WG1-WG3/$\overline{WG1}$-$\overline{WG3}$ (P6.0-P6.5)	3-phase output signals and their complements used in motor control applications. The pins can also be configured as standard output pins.
$\overline{WR}$/$\overline{WRL}$ (P5.2)	Write and Write Low output to external memory. $\overline{WR}$ will go low every external write. $\overline{WRL}$ will go low only for external writes to an even byte. Can be used as standard I/O when not used as $\overline{WR}$/$\overline{WRL}$.
XTAL1	Input of the oscillator inverter and the internal clock generator. This pin should be used when using an external clock source.
XTAL2	Output of the oscillator inverter.

Appendix B
MCS®-96
Instruction Set

MCS®-96 INSTRUCTION SET

OVERVIEW

This chapter of the manual gives a description of each instruction recognized by the MCS® -96 architecture. The instructions are sorted alphabetically by the assembly language mnemonic.

Additional information including instruction execution times and a description of the different addressing modes can be found in the following documents:

MCS-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048 (Intel systems)
Order Number 122351 (DOS systems)

MCS-96 UTILITIES USER'S GUIDE
Order Number 122049 (Intel systems)
Order Number 122356 (DOS systems)

PL/M-96 USER'S GUIDE
Order Number 122134 (Intel systems)
Order Number 122361 (DOS systems)

C-96 USER'S GUIDE
Order Number 167632

80C196KC USER'S GUIDE
Order Number 270704

80C196KB USER'S GUIDE
Order Number 270651

MCS-96 ARCHITECTURAL OVERVIEW
Order Number 270250

The instruction set descriptions in the following sections do not always show the effect on the program counter (PC). Unless otherwise specified, all instructions increment the PC by the number of bytes in the instruction.

A set of acronyms are used to make the instruction set descriptions easier to read, their definitions are listed below:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D".

baop. A byte operand which is addressed by any of the address modes.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D".

waop. A word operand which is addressed by any of the address modes.

Lreg. A 32-bit register in the internal register file.

cadd. An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminant state after the operation.

Generic Jumps and Calls. The assembler for the MCS-96 family provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user's guide should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

Indirect Shifts. The indirect shift operations use registers 24 through 255 (18H–0FFH), since 0–15 are direct operators and registers 16 through 23 are Special Function Registers. Note that indirect shifts through SFRs are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 5 least significant bits.

Direct Addressing

To use direct addressing, specify any label or expression as the operand. For **waop**, the operand specified must be on word boundary.

If the operand refers to a register, register addressing is used:

	baop		waop	
Pattern	xxxxxx00	breg	xxxxxx00	wreg
Additional Bytes:	**1**		**1**	
Additional Cycles:	**0**		**0**	

Otherwise, long indexing based on register 0 is used:

Pattern	xxxxxx11	00000001	loc-low	loc-high

Additional Bytes:		**3**
Additional Cycles:	**On-Chip Memory:**	**3**
	Off-Chip Memory:	**8**

Examples

```
PUSH ALPHA
PUSH GAMMA+1
```

Immediate Addressing

To use immediate addressing, specify a number (#), followed by the immediate expression. For **baop**, the expression must be in the range of −256 to +255. For **waop**, the expression may be in the range of −65536 to +65535.

	baop	
Pattern	xxxxxx01	value

Additional Bytes: 1
Additional Cycles: 0

or

	waop		
Pattern	xxxxxx01	val-low	val-high

Additional Bytes: 2
Additional Cycles: 1

Examples

```
CMPB  ALPHA,#2
PUSH  #GAMMA
```

Indirect Addressing

To use indirect addressing, specify a word register within square brackets. The value of that register specifies the address of the required operand. Since a word register has an even address, only seven bits are needed to store its address: the least significant bit is always zero. That bit in the instruction is used to distinguish between indirect addressing with or without auto increment.

For **waop**, the run-time value of that register must be even.

Pattern	xxxxxx10	wreg 0

Additional Bytes: 1
Additional Cycles: **On-Chip Memory:** 2
Off-Chip Memory: 7

Examples

```
LD    ALPHA,[BETA]
ADD   ALPHA,BETA,[GAMMA]
```

Indirect Addressing with Auto Increment

To use indirect addressing with auto increment, specify a word register within square brackets, followed by a + sign. The value of that register specifies the address of the required operand. After the operand is fetched, the register is incremented by 1 (for **baop**) or 2 (for **waop**).

Since a word register has an even address, only seven bits are needed to store its address: the least significant bit of the address is always zero. That bit in the instruction is used to distinguish between indirect addressing with or without auto increment, and is set to one.

For **waop**, the run-time value of that register must be even.

Pattern	xxxxxx10	wreg 1

Additional Bytes: 1
Additional Cycles: **On-Chip Memory:** 3
Off-Chip Memory: 8

Examples

```
CMPB  ALPHA,[BETA]+
SUB   ALPHA,BETA,[GAMMA]+
```

Based or Indexed Addressing

This addressing mode is used when part of the address is known at compile time, and a part is computed at run time. This is useful both for indexing into arrays, where the base address is known at compile time and the offset of the desired element is computed dynamically, and to reference based data structure, where the base address is dynamic, but the offset of the desired member in the structure is known at compile time.

To use based or indexed addressing, specify any label or expression as the offset, followed by a word register within square brackets. The specified offset is added to the value of that register to get the address of the required operand.

For **waop**, the offset must be even, and the run-time value of that register must be even.

Since a word register has an even address, only seven bits are needed to store its address: the least significant bit of the address is always zero. That bit in the instruction is set to zero to indicate short indexing and set to one to indicate long indexing.

If the offset is in the range of −128 to +127 inclusive, short indexing is used:

Pattern	xxxxxx11	wreg 0	offset

Additional Bytes: **2**
Additional Cycles: **On-Chip Memory:** **2**
Off-Chip Memory: **7**

Otherwise, long indexing is used:

Pattern	xxxxxx11	wreg 1	offs.low	offs.hi

Additional Bytes: 3
Additional Cycles: **On-Chip Memory:** 3
Off-Chip Memory: 8

Examples

```
LD    ALPHA,2[BETA]
CMP   ALPHA,BETA[GAMMA]
```

1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
          DST     SRC
ADD       wreg,   waop
```

Object Code Format: [011001aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

2. ADD (Three Operands) — ADD WORDS

Operation: The Sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
ADD       Dwreg,  Swreg,  waop
```

Object Code Format: [010001aa] [waop] [Swreg] [Dwreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
          DST     SRC
ADDB      breg,   baop
```

Object Code Format: [011101aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADDB	Dbreg,	Sbreg,	baop

Object Code Format: [010101aa] [baop] [Sbreg] [Dbreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDC	wreg,	waop

Object Code Format: [101001aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDCB	breg,	baop

Object Code Format: [101101aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
AND     wreg,   waop
```

Object Code Format: [011000aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
AND     Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa] [waop] [Swreg] [Dwreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
          DST     SRC
ANDB      breg,   baop
```

Object Code Format: [011100aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
ANDB      Dbreg,  Sbreg,  baop
```

Object Code Format: [010100aa] [baop] [Sbreg] [Dbreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

11. BMOV — BLOCK MOVE

Operation: This instruction is used to move a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with auto-increment addressing modes. A long register addresses the source and destination pointers which are stored in adjacent word registers. The number of transfers is specified by a word register. The blocks of data can reside anywhere in memory, but should not overlap.

```
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS) + 2) ← DSTPTR + 2
COUNT ← COUNT − 1
if COUNT <> 0 then go to LOOP
```

PTRS CNTREG

Assembly Language Format: BMOV Lreg, wreg

Object Code Format: [11000001] [wreg] [Lreg]

Flags Affected					
Z	N	C		VT	ST
—	—	—	—	—	—

NOTES:

1. CNTREG does not get decremented during the instruction.
2. It is easy to unintentionally create a very long un-interruptable operation with this instruction.

To provide an interruptable version of the BMOV for large blocks, the BMOV instruction can be used with the DJNZ instruction. This is possible because the pointers are modified, but CNTREG is not. Consider the example:

```
        LD      PTRS, SRC           ;Pointer to base of sources table
        LD      PTRS+2, DST         ;Pointer to base of destination table
        LD      CNTREG, #COUNT      ;Number of bytes to move per set
        LD      CNTSET, #SETS       ;Number of sets to move
Move:   BMOV    PTRS, CNTREG        ;Move one set
        DJNZ    CNTSET, MOVE        ;Decrement set counters and move again
```

12. BMOVI — INTERRUPTABLE BLOCK MOVE

Operation: This instruction is used to move a block of word data from one location in memory to another and is interruptable. The source and destination addresses are calculated using the indirect with auto-increment addressing modes. A long register addresses the source and destination pointers which are stored in adjacent word registers. The number of transfers is specified by a word register. The blocks of data can reside anywhere in memory, but should not overlap.

```
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS) + 2) ← DSTPTR + 2
COUNT ← COUNT − 1
if COUNT <> 0 then go to LOOP
```

PTRS CNTREG

Assembly Language Format: BMOVI Lreg, wreg

Object Code Format: [11001101] [wreg] [Lreg]

Flags Affected					
Z	N	C		VT	ST
—	—	—	—	—	—

NOTES:

1. CNTREG does not get decremented during the instruction. However, if the BMOVi is interrupted, CNTREG will be updated. Therefore, CNTREG must be reloaded before starting a BMOVi.

13. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

14. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

15. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.
(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001] [breg]

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

16. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.
C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Flags Affected					
Z	N	C	V	VT	ST
—	—	0	—	—	—

17. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

18. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
CMP     wreg,   waop
```

Object Code Format: [100010aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
CMPB    breg,   baop
```

Object Code Format: [100110aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

20. CMPL — COMPARE LONG (80C196KB and 80C196KC only)

Operation: This instruction is used to compare the magnitudes of two double word (long) operands. The operands are specified using the direct addressing mode. Five PSW flags are set following this operation, but the operands are not affected.

DST-SRC

Assembly Language Format:

```
        DST SRC
CMPL    Lreg, Lreg
```

Object Code Format: [11000101] [src Lreg] [dst#Lreg]

Flags Affected					
Z	N	V	VT	C	ST
✓	✓	✓	✓	✓	—

21. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

22. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

23. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.
Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

24. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIV	lreg,	waop

Object Code Format: [11111110] [100011aa] [waop] [lreq]

Flags Affected						
Z	N	C	V	VT	ST	
—	—	—	?	↑	—	8096BH
—	—	—	✓	↑	—	80C196KB, 80C196KC

25. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVB	wreg,	baop

Object Code Format: [11111110] [100111aa] [baop] [wreg]

Flags Affected						
Z	N	C	V	VT	ST	
—	—	—	?	↑	—	8096BH
—	—	—	✓	↑	—	80C196KB, 80C196KC

26. DIVU — DIVIDE WORDS

Operation: This instruction divides the content of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order WORD will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVU    lreg,   waop
```

Object Code Format: [100011aa] [waop] [lreq]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↙	↑	—

27. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination, (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVUB   wreg,   baop
```

Object Code Format: [100111aa] [baop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↙	↑	—

28. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) <> 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000] [breg] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

29. DJNZW — DECREMENT AND JUMP IF NOT ZERO WORD (80C196KB and 80C196KC only)

Operation: This instruction is the same as the DJNZ except that the count is a word operand. A counter word is decremented; if the result is not zero the jump is taken. The range of the jump is −128 to +127.

```
COUNT ← COUNT ← − 1
if COUNT <> 0 then
PC ← PC + disp (sign extended)
```

Assembly Language Format: DJNZW wreg,cadd

Object Code Format: [11100001] [wreg] [disp]

Flags Affected					
Z	N	V	VT	C	ST
—	—	—	—	—	—

30. DPTS — DISABLE PERIPHERAL TRANSACTION SERVER (PTS)

Operation: The PTS is disabled following the execution of this instruction.

PTS Disable (PSW.10) ← 0

Assembly Language Format: DPTS

Object Code Format: [11101100]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

31. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9) ← 1

Assembly Language Format: EI

Object Code Format: [11111011]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

32. EPTS — ENABLE PERIPHERAL TRANSACTION SERVER (PTS)

Operation: The PTS is enabled following the execution of this instruction.
PTS Enable (PSW.10) ← 1

Assembly Language Format: EPTS

Object Code Format: [11101101]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

33. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST) < 8000H then
  (high word DEST) ← 0
else
  (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

34. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST) < 80H then
  (high byte DEST) ← 0
else
  (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

35. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

36. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.
(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

37. IDLPD — IDLE/POWERDOWN

Operation: This instruction is used for entry into the idle and powerdown modes. Selecting IDLE or POWERDOWN is done using the key operand. If the operand is not a legal key, the part executes a reset sequence. The bus controller will complete any prefetch cycle in progress before the CPU stops or resets.

```
if KEY = 1 then enter IDLE
else if KEY = 2 then enter
POWERDOWN
else execute reset.
```

Assembly Language Format: IDLPD #key (key is 8-bit value)

Object Code Format: [11110110] [key]

	Flags Affected					
	Z	N	V	VT	C	ST
Legal Key:	—	—	—	—	—	—
Illegal Key:	0	0	0	0	0	0

(— = Unchanged)

38. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
  PC ← PC + disp (sign-extened to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb] [breg] [disp]
where bbb is the bit number within the specified register.

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

39. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg] [disp]
where bbb is the bit number within the specified register.

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

40. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

41. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

42. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

43. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

44. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

45. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

46. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

47. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

48. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111 [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

49. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If the carry flag is set (i.e., 1) and the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

50. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

51. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

52. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

53. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

54. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

55. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow trap flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100] [disp]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

56. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP − 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111] [disp-low] [disp-hi]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

57. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST    SRC
LD    wreg,  waop
```

Object Code Format: [101000aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

58. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST    SRC
LDB   breg,  baop
```

Object Code Format: [101100aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

59. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
    (high byte DEST) ← 0
else
    (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
          DST    SRC
LDBSE     wreg,  baop
```

Object Code Format: [101111aa] [baop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

60. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
          DST    SRC
LDBZE     wreg,  baop
```

Object Code Format: [101011aa] [baop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

61. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111] [disp-low] [disp-hi]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

62. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MUL     lreg,   waop
```

Object Code Format: [11111110] [011011aa] [waop] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

63. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MUL     lreg,   wreg,   waop
```

Object Code Format: [11111110] [010011aa] [waop] [wreg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

64. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULB    wreg,   baop
```

Object Code Format: [11111110] [011111aa] [baop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

65. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MULB    wreg,   breg    baop
```

Object Code Format: [11111110] [010111aa] [baop] [breg] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

66. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULU    lreg,   waop
```

Object Code Format: [011011aa] [waop] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

67. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULU	lreg,	wreg,	waop

Object Code Format: [010011aa] [waop] [wreg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

68. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULUB	wreg,	baop

Object Code Format: [011111aa] [baop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

69. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULUB	wreg,	breg,	baop

Object Code Format: [010111aa] [baop] [breg] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

70. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

71. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.
(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

72. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

73. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111] [breg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	?	0	—	—	—

74. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT (DEST)
```

Assembly Language Format: NOT wreg

Object Code Format: [00000010] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

75. NOTB — COMPLEMENT BYTE

Operation: The vaule of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

76. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

	DST	SRC
OR	wreg,	waop

Object Code Format: [100000aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

77. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

78. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa] [waop]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

79. POPA — POP ALL (80C196KB and 80C196KC only)

Operation: This instruction is used instead of POPF to support the 8 additional interrupts. It is similar to POPF, but pops two words instead of one. The first word is popped into the INT_MASK1/WSR register pair, while the second word is popped into the PSW/INT_MASK register pair. As a result of this instruction the SP is incremented by 4. Interrupts cannot occur between this instruction and the one following it.

```
INT_MASK1/WSR ← (SP)
SP ← SP + 2
PSW/INT_MASK ← (SP)
SP ← SP + 2
```

Assembly Language Format: POPA

Object Code Format: [11110101]

Flags Affected							
Z	N	V	VT	C	X	I	ST
✓	✓	✓	✓	✓	X	✓	✓

(✓ = changed)

80. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

```
(PSW) ← (SP)
SP ← SP + 2
```

Assembly Language Format: POPF

Object Code Format: [11110011]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

81. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

```
SP ← SP − 2
(SP) ← (DEST)
```

Assembly Language Format: PUSH waop

Object Code Format: [110010aa] [waop]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

82. PUSHA — PUSH ALL

Operation: This instruction is used instead of PUSHF to support the 8 additional interrupts. It is similar to PUSHF, but pushes two words instead of one. The first word pushed is the same as for the PUSHF instruction, PSW/INT_MASK. The second word pushed is formed by the INT_MASK1/WSR register pair. As a result of this instruction the PSW, INT_MASK, and INT_MASK1 registers are cleared, and the SP is decremented by 4. Interrupts are disabled in two ways by this instruction since both PSW.9 and the interrupt masks are cleared. Interrupts cannot occur between this instruction and the one following it.

```
SP ← SP − 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
SP ← SP − 2
(SP) ← INT_MASK1/WSR
INT_MASK1 ← 0
```

Assembly Language Format: PUSHA

Object Code Format: [11110100]

Flags Affected							
Z	N	V	VT	C	X	I	ST
0	0	0	0	0	X	0	0

83. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

```
SP ← SP − 2
(SP) ← PSW
PSW ← 0
```

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

84. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

```
PC ← (SP)
SP ← SP + 2
```

Assembly Language Format: RET

Object Code Format: [11110000]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

85. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value. Executing this instruction will cause a pulse to appear on the reset pin.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

86. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP − 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx] [disp-low]

where xxx holds the three high-order bits of displacement.

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

87. SETC — SET CARRY FLAG

Operation: The carry flag is set.

```
C ← 1
```

Assembly Language Format: SETC

Object Code Format: [11111001]

Flags Affected					
Z	N	C	V	VT	ST
—	—	1	—	—	—

88. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
        SHL     wreg,#count
or
        SHL     wreg,breg
```

Object Code Format: [00001001] [cnt/breg] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
↙	?	↙	↙	↑	—

89. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    TEMP ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHLB    breg,#count
or
        SHLB    breg,breg
```

Object Code Format: [00011001] [cnt/breg] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

90. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHLL    lreg,#count
or
        SHLL    lreg, breg
```

Object Code Format: [00001101] [cnt/breg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

91. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHR     wreg,#count
or
        SHR     wreg,breg
```

Object Code Format: [00001000] [cnt/breg] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

92. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRA    wreg,#count
or
        SHRA    wreg,breg
```

Object Code Format: [00001010] [cnt/breg] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

93. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAB   breg, #count
or
        SHRAB   breg,breg
```

Object Code Format: [00011010] [cnt/breg] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

94. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAL   lreg,#count
or
        SHRAL   lreg,breg
```

Object Code Format: [00001110] [cnt/breg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

95. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRB    breg,#count
or
        SHRB    breg,breg
```

Object Code Format: [00011000] [cnt/breg] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

96. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register above 15H. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DSET) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRL    lreg,#count
or
        SHRL    lreg,breg
```

Object Code Format: [00001100] [cnt/breg] [lreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

97. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx] [disp-low]
where xxx holds the three high order bits of the displacement.

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

98. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000] [breg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

99. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

```
        SRC     DST
ST      wreg,   waop
```

Object Code Format: [110000aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

100. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

```
        SRC     DST
STB     breg,   baop
```

Object Code Format: [110001aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

101. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUB     wreg,   waop
```

Object Code Format: [011010aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

102. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2,
SUB     wreg,   wreg,   waop
```

Object Code Format: [010010aa] [waop] [Sweg] [Dwreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

103. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUBB    breg,   baop
```

Object Code Format: [011110aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

104. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the second byte operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
SUBB    breg,   Sbreg   baop
```

Object Code Format: [010110aa] [baop] [Sbreg] [Dbreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

105. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the orignal destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
        DST     SRC
SUBC    wreg,   waop
```

Object Code Format: [101010aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

106. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
        DST     SRC
SUBCB   breg,   baop
```

Object Code Format: [101110aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

107. TIJMP — TABLE INDIRECT JUMP (80C196KC and 80C196KR only)

Operation: The execution continues at an address selected out of a table of addresses. TBASE is a word register which contains the 16-bit address of the beginning of the table. INDEX is a word register containing the 16-bit address of a byte which contains the index into the table. INDEX_MASK is ANDed with the index. The index must be between 0 and 128.

ADDRESS CALCULATION

```
[INDEX] AND INDEX_MASK = OFFSET
          (2 * OFFSET) + [TBASE] = DEST X
```

Assembly Language Format:

```
        TBASE [INDEX] #INDEX_MASK
TIJMP wreg,   wreg     #byte
```

Object Code Format:

```
                [INDEX]                 [TBASE]
[  11100010  ] [  wreg  ] [  #byte  ] [  wreg  ]
```

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

108. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

```
SP ← SP – 2
(SP) ← PC
PC ← (2010H)
```

Assembly Language Format: This instruction is not supported by revision 1.2 of the 8096 assembly language.

Object Code Format: [11110111]

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

109. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XOR     wreg,   waop
```

Object Code Format: [100001aa] [waop] [wreg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

110. XCH — EXCHANGE WORD

Operation: The value of the source (rightmost) word operand is exchanged with the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
        DST     SRC
XCH     wreg,   waop
```

Object Code Format:

```
[ 00000100 ] [ waop ] [ wreg ]
[ 00001011 ] [ waop ] [ wreg ]
```

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

111. XCHB — EXCHANGE BYTE

Operation: The value of the source (rightmost) byte operand is exchanged with the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
        DST     SRC
XCHB    breg,   baop
```

Object Code Format:

```
[ 00010100 ] [ baop ] [ breg ]
[ 00011011 ] [ baop ] [ breg ]
```

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

112. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XORB    breg,   baop
```

Object Code Format: [100101aa] [baop] [breg]

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—